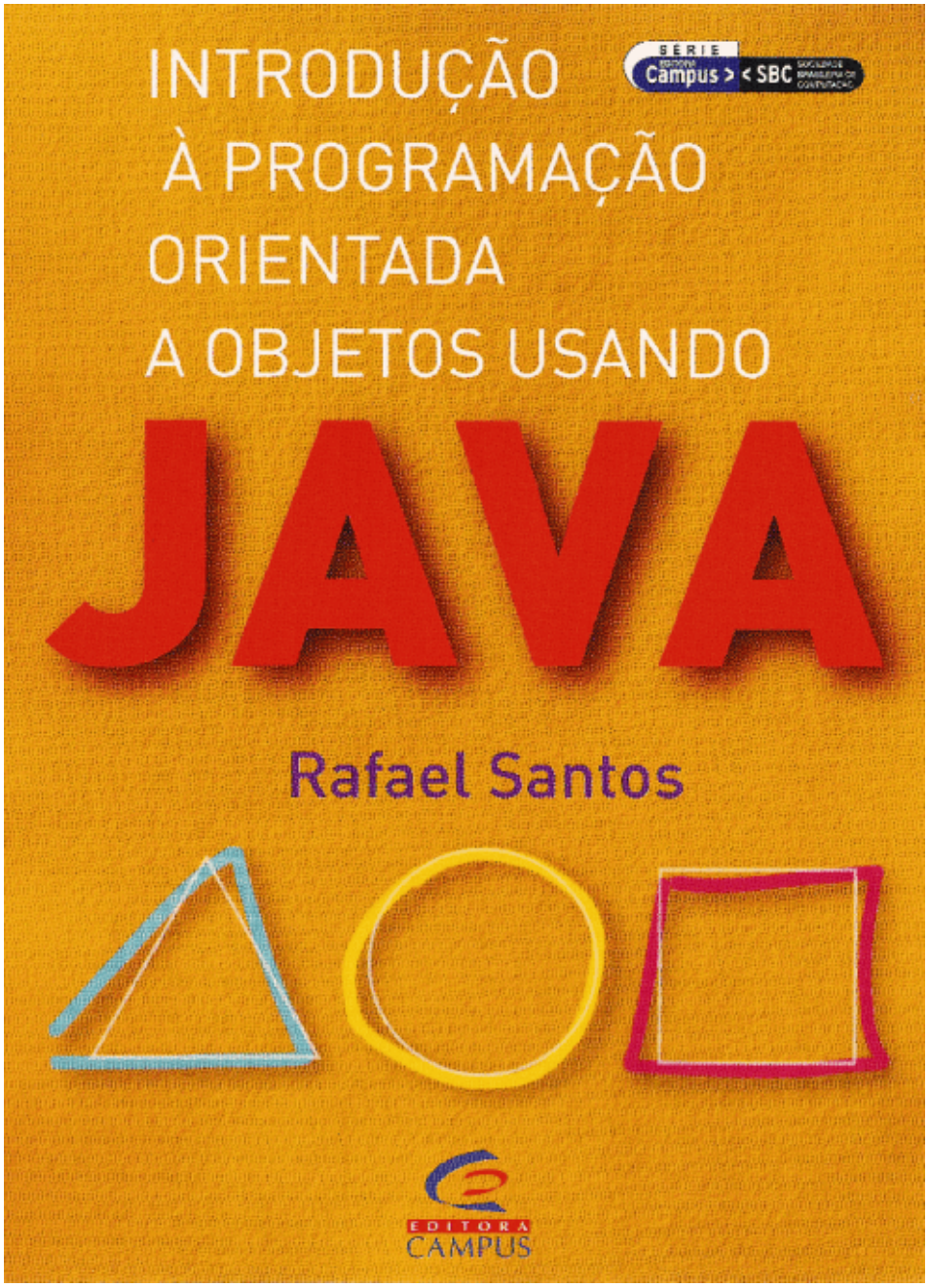


Lista de Exercícios

27 de Maio de 2003

Prefácio

Sobre este documento

Este documento é a lista de exercícios do livro *'Introdução à Programação Orientada a Objetos*, ISBN 85-352-1206-X, Editora Campus. Este documento contém **570** exercícios correspondentes aos tópicos abordados no livro.

Este material é disponibilizado **gratuitamente** no site http://www.campus.com.br. Cópias também podem ser obtidas no endereço http://www.directnet.com.br/users/rafael.santos.

Este material está sujeito a modificações sem aviso prévio. Em particular, é bem possível que mais exercícios sejam incluídos no material. Por favor, verifique regularmente os sites de distribuição. A versão do documento é mostrada na sua capa (data de modificação), e uma lista de modificações significativas é mostrada a seguir.

Histórico de modificações

- **27/05/2003** Primeira versão para disponibilização (**570** exercícios)

Perguntas e respostas sobre os exercícios do livro

Durante a criação deste livro, algumas perguntas, críticas e comentários surgiram, freqüentemente feitas por colegas e estudantes dos cursos para os quais usei este material. Algumas dessas perguntas são apresentadas aqui, com as respectivas respostas. Algumas das perguntas não chegaram a ser feitas, mas é possível que acabem surgindo depois da publicação, sendo respondidas a seguir.

O número de exercícios não é excessivo, especialmente considerando que muitos são tão similares que chegam a ser repetitivos? Sim, mas isso é considerado uma característica positiva da lista de exercícios: o estudante pode escolher quais dos exercícios aparentemente similares vai resolver, e deduzir a solução dos outros. Leitores que estudem em grupos podem também dividir tarefas e comparar resultados usando exercícios similares. Espero que a seleção ampla também facilite a um instrutor passar trabalhos, dividir tarefas etc.

Não existem exercícios resolvidos? Quase sempre que um conceito é introduzido, uma figura contendo uma classe ou aplicação em Java é mostrada. Considero que isso equivale a exercícios resolvidos, do tipo "Demonstre o conceito descrito anteriormente com uma classe ou aplicação". Muitos dos exercícios propostos, em especial os mais básicos, pedem modificações simples de listagens apresentadas – esses exercícios são considerados como parcialmente resolvidos.

Sobre estes exercícios

Os exercícios apresentados para cada capítulo são divididos em cinco categorias de dificuldade, marcadas com um número de estrelas correspondente:

- **Uma estrela (⋆)**: Exercícios teóricos ou práticos que podem ser resolvidos rapidamente, geralmente através de consultas a programas mostrados ou respostas de outros exercícios, e modificação de exemplos ou outras respostas. Em muitos casos podem ser resolvidos mentalmente, isto é, sem a necessidade de escrever, compilar e executar programas. Existem **191** exercícios desta categoria no livro.

- **Duas estrelas (⋆⋆)**: Exercícios teóricos ou práticos que exigem um pouco mais de raciocínio e modificações ligeiras de exemplos apresentados. Exercícios desta categoria geralmente precisarão ser compilados e executados para verificação dos resultados. Existem **169** exercícios desta categoria no livro.
- **Três estrelas (⋆⋆⋆)**: Exercícios que requerem compreensão mais completa dos conceitos envolvidos, mas mesmo assim podem ser resolvidos com base em exemplos ou respostas a outros exercícios. Existem **151** exercícios desta categoria no livro.
- **Quatro estrelas (⋆⋆⋆⋆)**: Exercícios que requerem compreensão ainda mais completa e profunda dos conceitos, e que geralmente exigem um conhecimento razoável de algoritmos e técnicas de programação. Existem **41** exercícios desta categoria no livro.
- **Cinco estrelas (⋆⋆⋆⋆⋆)**: Exercícios que requerem a solução de um problema mais completo e complexo, envolvendo vários conceitos diferentes da disciplina. Estes exercícios podem servir como base para projetos mais interessantes. Existem **15** exercícios desta categoria no livro.

Muitos dos exercícios são aparentemente repetitivos e até mesmo redundantes. Isso é proposital: se o estudante conseguir resolver um dos exercícios aparentemente repetidos, poderá deduzir a solução de outros mais facilmente. A criação de vários exercícios aparentemente redundantes também facilita ao professor a criação de listas de exercícios distintas.

Sumário

Lista de Figuras

Capítulo 1

Introdução à programação orientada a objetos

1.1 Exercícios do Capítulo 1

Os exercícios deste capítulo são puramente teóricos, e sua solução não requer conhecimentos da sintaxe da linguagem Java. Para melhor resultado, as soluções encontradas devem ser discutidas com outros estudantes.

Para os exercícios relacionados com criação de modelos, é sugerida a criação também dos diagramas mostrados nas seções 1.5.1 a 1.5.4.

Exercício 1.1: ⋆

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

Exercício 1.2: ⋆

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

Exercício 1.3: ⋆

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

Exercício 1.4: ⋆

Escreva um modelo para representar uma lâmpada que está à venda em um supermercado. Que dados 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` (figura 1.3) 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` (figura 1.3), uma operação `estáLigada` que retorne verdadeiro se a lâmpada estiver ligada e falso, caso contrário.

Exercício 1.8: ⋆

A operação `abreConta` do modelo `ContaBancariaSimplificada` (figura 1.5) 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` (figura 1.5) 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` (figura 1.7) crie o modelo `HoraAproximada`, que represente uma hora qualquer (usando valores para representar horas e minutos). Que dados e operações este modelo deve ter?

Exercício 1.11: ⋆

Baseado no modelo `Data` (figura 1.7) 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 dados e operações este modelo deve ter? Que dados 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` (figura 1.7) 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: ⋆

A operação `inicializaRegistro` do modelo `RegistroAcademico` (figura 1.9) não verifica nenhum dos dados passados, simplesmente copiando-os para os valores do modelo. Modifique essa operação para que, se o ano de matrícula for menor do que 1990, seja considerado como sendo zero (inválido).

Exercício 1.15: ⋆

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

Exercício 1.16: ⋆

Usando o resultado do exercício 1.15 como base, crie um modelo `LivroDeLivraria` que represente os dados básicos de um livro que está à venda em uma livraria. Veja também o exercício 1.17.

Exercício 1.17: ⋆

Usando o resultado do exercício 1.15 como base, crie um modelo `LivroDeBiblioteca` que represente os dados básicos de um livro de uma biblioteca, que pode ser emprestado a leitores. Veja também o exercício 1.16.

Exercício 1.18: ★

Usando o resultado do exercício 1.15 como base, crie um modelo `DicionarioBilingue` que represente os dados básicos de um dicionário de línguas (por exemplo, português-inglês, latim-aramaico etc.).

Exercício 1.19: ★

Usando o resultado do exercício 1.15 como base, crie um modelo `LivroComSoftware` que represente os dados básicos de um livro acompanhado de software. *Dica:* O software pode ser demonstrativo ou não e o título pode ser diferente do título do livro.

Exercício 1.20: ★

Crie um modelo para representar um professor de uma disciplina qualquer. *Dica:* Use, para orientação, o modelo `RegistroAcademico` (figura 1.9).

Exercício 1.21: ★

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

Exercício 1.22: ★

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

Exercício 1.23: ★

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 1.24: ★

Crie um modelo `Veiculo` que represente os dados e operações de um veículo de transporte como nome, número de passageiros que pode carregar, tipo (aéreo, terrestre ou aquático), número de rodas etc.

Exercício 1.25: ★

Com base no exercício 1.24, crie um modelo `VeiculoTerrestre` que tenha os mesmos dados do modelo `Veiculo`. Quais serão as principais diferenças entre o modelo criado nesse exercício e o modelo `Veiculo`?

Exercício 1.26: ★

Com base nos exercícios 1.24 e 1.25, crie um modelo `AutomovelDePasseio` que tenha os mesmos dados do modelo `VeiculoTerrestre`. Quais serão as principais diferenças entre o modelo criado nesse exercício e o modelo `VeiculoTerrestre`?

Exercício 1.27: ★

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

Exercício 1.28: ★★

Modifique a operação `mostraData` no modelo `Data` (figura 1.7) para que o mês seja mostrado por extenso. *Dica:* Veja o exercício 1.13.

Exercício 1.29: ★★

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 1.27) para que dois valores de horas trabalhadas e dois valores de salário-hora sejam usados.

Exercício 1.30: ★★

Modifique a operação `calculaSalárioMensal` no modelo `Empregado` (veja o exercício 1.27) para que todos os empregados do departamento Diretoria tenham 10% de bônus salarial.

Exercício 1.31: ★★

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

Exercício 1.32: ★★

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

Exercício 1.33: ★★

Crie um modelo para representar uma linha, unida por dois pontos no espaço cartesiano de duas dimensões, usando o modelo criado no exercício 1.23. Que dados e operações esse modelo deve ter?

Exercício 1.34: ★★

Crie um modelo para representar um retângulo, cujos pontos opostos sejam instâncias do modelo `Ponto2D` (exercício 1.23). Veja também o exercício 1.33.

Exercício 1.35: ★★

Considerando o exercício 1.21, escreva um modelo `Partida` que represente uma partida entre dois times do esporte. Como seria a operação `imprimeVencedor` desse modelo?

Exercício 1.36: ★★

Escreva um modelo que represente um polígono regular de até dez lados. Que dados 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 1.37: ★★★

Considere o modelo `Lampada` mostrado em pseudocódigo na figura 1.3. 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 1.38: ★★★

Suponha que uma determinada escola ofereça três tipos de bolsa para alguns de seus estudantes: 30% de desconto, 70% de desconto e bolsa integral com 100% de desconto. Que modificações deveriam ser feitas no modelo `RegistroAcademico` (figura 1.9)?

Exercício 1.39: ★★★

Usando o exercício 1.38 como base, considere que a escola também tenha cursos diurnos e noturnos, e ofereça para todos os estudantes, indiferente de terem ou não bolsa, desconto de 20% no turno da manhã. Que modificações deveriam ser feitas no modelo `RegistroAcademico` (figura 1.9)? *Dica:* Um novo dado para representar o turno de estudos (diurno ou noturno) deverá ser criado e manipulado.

Exercício 1.40: ★★★

Crie um modelo `CDDeMusicas` que contenha várias instâncias do modelo `Musica` (exercício 1.22). Como você acha que podemos fazer para representar, em um `CDDeMusicas`, um número variável de instâncias de `Musica`?

Exercício 1.41: ★★★

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 1.42: ★★★★

A operação `inicializaData` do modelo `Data` (figura 1.7) 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

2.1 Exercícios do Capítulo 2

Para os exercícios relacionados com criação de classes, é sugerido que se usem os modificadores de acesso adequados a cada campo e método, tentando fazer com que os campos sejam sempre protegidos para acesso e que os métodos sejam públicos quando devido. Para isso, deve-se tentar escrever ao menos métodos para inicializar ou modificar os campos e imprimir os valores destes.

Para praticar, também é sugerida a criação dos diagramas mostrados nas seções 1.5.1 a 1.5.4 e a inclusão de comentários nas classes, métodos e trechos adequados.

Exercício 2.1: ⋆

Quais dos identificadores abaixo podem ser usados como nomes de classes, campos, 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, campos, 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, campos, 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, 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: ★

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

```
class DoisValores
  {
  /**
   * Declaração dos campos desta classe
   */
  int valor1,valor2;
  /**
   * Declaração dos métodos desta classe
   */
  int maior()
    {
    if (valor1 > valor2)
      return true;
    else return false;
    }
  void menor()
    {
    if (valor1 < valor2)
      return valor1;
    else return valor2;
    }
  } // fim da classe
```

Exercício 2.8: ★

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

```
class Registro De Eleitor
  {
  /**
   * Declaração dos campos desta classe
   */
  int tituloDeEleitor; // número do título do eleitor
  String nome; // nome do eleitor
  short zonaEleitoral; // número da zona eleitoral
  } // fim da classe
```

Exercício 2.9: ★

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

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

Exercício 2.10: ★

Escreva a classe `Lampada` correspondente ao modelo da figura 1.3. Que tipo de dado pode ser usado para representar o campo `estado`?

Exercício 2.11: ★

Escreva na classe `Lampada` (veja o exercício 2.10) o método correspondente à resposta do exercício 1.7.

Exercício 2.12: ★
Modifique a resposta do exercício 2.10 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 2.13: ★
Modifique o método `mostraData` da classe `Data` (figura 2.7) para que o mês seja mostrado por extenso em vez de numeral (isto é, quando o mês for 3, que o método imprima março, etc.)

Exercício 2.14: ★
Escreva a classe `Ponto2D`, correspondente ao modelo da resposta do exercício 1.23.

Exercício 2.15: ★
Escreva a classe `Veiculo`, correspondente ao modelo da resposta do exercício 1.24.

Exercício 2.16: ★
Escreva a classe `VeiculoTerrestre`, correspondente ao modelo da resposta do exercício 1.25. Veja também o exercício 2.15.

Exercício 2.17: ★
A classe `Data` (figura 2.7) pode representar datas não-válidas, com os valores de dia, mês e ano iguais a zero. Modifique o método `mostraData` para que, se a data encapsulada não for válida, uma mensagem "`Data Inválida`" seja impressa, em vez dos valores de dia, mês e ano (veja o exercício 1.13).

Exercício 2.18: ★
Escreva a classe `LampadaTresEstados` correspondente à resposta do exercício 1.5. Que tipo de dado pode ser usado para representar o campo `estado`?

Exercício 2.19: ★
Escreva a classe `Lampada100Estados` correspondente à resposta do exercício 1.6. Considere também a resposta do exercício 2.18. Que tipo de dado pode ser usado para representar o campo `estado`?

Exercício 2.20: ★
Escreva uma classe `HoraAproximada` que represente o modelo do exercício 1.10.

Exercício 2.21: ★
Usando o exercício 2.20 como referência, escreva uma classe `HoraPrecisa` que represente o modelo do exercício 1.11.

Exercício 2.22: ★
Escreva uma classe `Livro` que represente o modelo do exercício 1.15.

Exercício 2.23: ★
Escreva uma classe `LivroLivraria` que represente o modelo do exercício 1.16.

Exercício 2.24: ★
Escreva uma classe `LivroBiblioteca` que represente o modelo do exercício 1.17.

Exercício 2.25: ★

Modifique o método `éIgual` da classe `Data` (figura 2.7) para que uma data inválida seja considerada sempre diferente de qualquer outra.

Exercício 2.26: ★

O que aconteceria se todos os métodos da classe `Data` (figura 2.7) fossem declarados com o modificador `private`?

Exercício 2.27: ★

Escreva uma classe `AutomovelUsado` que represente os dados de um automóvel usado à venda, como ano, modelo, quilometragem rodada, combustível, preço pedido etc. Que campos e métodos essa classe deve ter?

Exercício 2.28: ★

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

Exercício 2.29: ★★

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 através de métodos que devem zerar, incrementar e imprimir o valor do contador.

Exercício 2.30: ★★

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 2.29). Em que método esse campo deve ser modificado?

Exercício 2.31: ★★

Escreva a classe `RegistroAcademico`, baseada na classe `RegistroAcademicoSimples` (figura 2.2), fazendo com que todos os campos sejam privados e adicionando os métodos necessários ao funcionamento da classe.

Exercício 2.32: ★★

Considere os exercícios 2.29 e 2.30. Faça com que o contador que conta quantas vezes uma lâmpada foi acesa seja uma instância da classe `Contador`.

Exercício 2.33: ★★

Crie a classe `DataHora` que represente simultaneamente uma data e uma hora aproximada. *Dica:* O modelo pode conter instâncias das classes `Data` e `HoraAproximada`. Use os exercícios 1.12 e 2.20 como referência.

Exercício 2.34: ★★

Escreva a classe `Empregado`, correspondente à resposta do exercício 1.27.

Exercício 2.35: ★★

Crie uma classe `Linha` para representar uma linha, unida por dois pontos no espaço cartesiano de duas dimensões, usando duas instâncias da classe `Ponto2D`, criada no exercício 2.14. Veja também o exercício 1.33.

Exercício 2.36: ★★

Crie uma classe `Retangulo` para representar um retângulo cujos pontos opostos sejam duas instâncias da classe `Ponto2D`, que deve ter sido criada no exercício 2.14. Veja também o exercício 1.34.

Exercício 2.37: ★★
Escreva a classe `PoligonoRegular`, correspondente à resposta do exercício 1.36.

Exercício 2.38: ★★
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 campos encapsulados.

Exercício 2.39: ★★★
Escreva uma classe `ContaBancariaSimplificada` que corresponda ao modelo na figura 1.5. Considere que modificadores de acesso devam ser usados para os métodos e campos da classe.

Exercício 2.40: ★★★
Se os métodos `abreConta`, `deposita` e `retira` que devem ter sido criados no exercício 2.39 forem criados como o modelo da figura 1.5 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 2.41: ★★★
Modifique a classe `Lampada` de acordo com o pedido no exercício 1.37.

Exercício 2.42: ★★★
Implemente a lógica correta de cálculo de anos bissextos e dias nos meses mostrada no exercício 1.42 na classe `Data`.

Exercício 2.43: ★★★
Escreva uma classe `ModeloDeComputador` que encapsule valores que definam a configuração de um microcomputador (tipo de processador, memória RAM, tamanho do disco rígido, tamanho do monitor, por exemplo). Essa classe deve ter um método `calculaPreço` que calcule o preço do computador como sendo a soma do custo de seus componentes:

- Placa-mãe: R$800
- Opções de processadores: 600Mhz a R$700, 800Mhz a R$830, 933Mhz a R$910
- Opções de memória: 128, 256, 384 ou 512 Mb, cada 128Mb custa R$350.
- Opções de disco rígido: 20 Gb a R$300, 40 Gb a R$420, 60 Gb a R$500.
- Opções de monitor: 15 polegadas a R$320, 17 polegadas a R$520.

Exercício 2.44: ★★★
Modifique a classe `Retangulo` (exercício 2.36) para que esta contenha métodos para retornar a área e o perímetro do retângulo encapsulado. *Dica:* A classe `Ponto2D`, que tem duas instâncias como campos na classe `Retangulo`, deve ter seus campos privados e métodos que permitam o acesso aos valores dos campos.

Exercício 2.45: ★★★
Modifique a classe `RegistroAcademico` (veja o exercício 2.31), adicionando o campo `códigoDoCurso`. Modifique também o método que imprime os valores dos campos da classe para que este imprima o *nome* do curso em vez do código. Invente vários códigos (números de dois dígitos) que representam diferentes cursos.

Exercício 2.46: ★★★
Implemente a solução do exercício 1.38 na classe `RegistroAcademico`. Veja também o exercício 2.31.

Exercício 2.47: ★★★
Implemente a solução do exercício 1.39 na classe `RegistroAcademico`. Veja também o exercício 2.31.

Exercício 2.48: ★★★

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` (figura 2.7) 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 2.49: ★★★★

Escreva em Java uma classe `RestauranteCaseiro` que implemente o modelo descrito na figura 1.1 da seção 1.2. Para isso, crie também uma classe `MesaDeRestaurante` que represente uma mesa de restaurante conforme mostrado na figura 1.1. Algumas sugestões sobre a criação dessas classes são:

- A classe `MesaDeRestaurante` deve ter campos 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.
- A classe `RestauranteCaseiro` deve ter várias campos que são instâncias da classe `MesaDeRestaurante`, para representar suas mesas.
- 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.

Exercício 2.50: ★★★★

Modifique a classe `Retangulo` (exercício 2.36) para que esta contenha dois métodos adicionais: um para verificar se uma instância da classe `Ponto2D` passada como argumento está localizada dentro da instância da classe `Retangulo`, que deverá retornar `true` se o ponto estiver dentro do retângulo, e outro para fazer o mesmo com uma instância da classe `Linha`. *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 2.1, 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. Para verificar se uma linha está dentro ou fora do retângulo, basta verificar os dois pontos que formam suas extremidades: somente se os dois pontos estiverem dentro do retângulo, a linha também estará: na figura 2.1, a linha $L2$ está dentro do retângulo, as linhas $L1$ e $L3$, não.
Veja também os exercícios 2.14, 2.35 e 2.44.

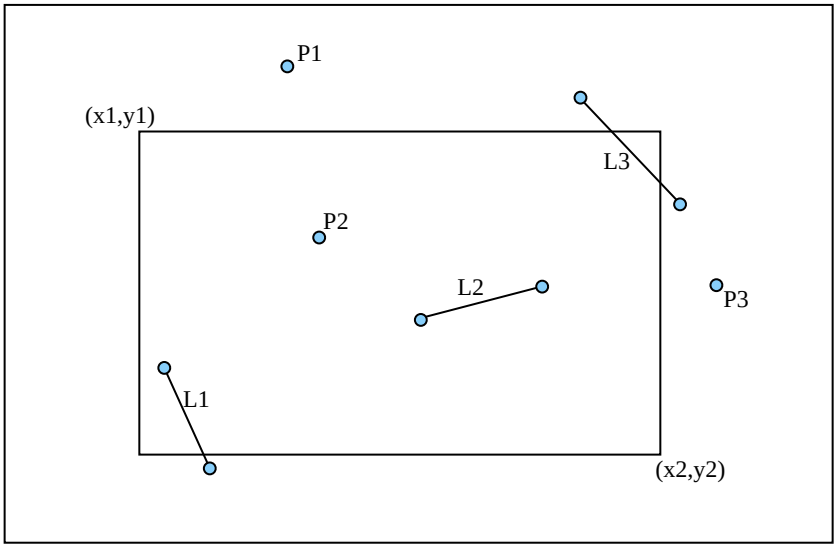


Figura 2.1: Pontos, linhas e retângulos no espaço cartesiano bidimensional.

Exercício 2.51: ★★★★

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

3.1 Exercícios do Capítulo 3

Exercício 3.1: ⋆

Explique, com suas palavras, por que uma classe como a `Ponto2D` (figura 3.4) não pode ser executada.

Exercício 3.2: ⋆

Escreva um programa em Java que imprima o seu nome.

Exercício 3.3: ⋆

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 3.4: ⋆

Escreva um programa em Java que use várias instâncias da classe `Lampada` (veja o exercício 2.10).

Exercício 3.5: ⋆

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

```
1  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 3.6: ⋆

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

```
1  public static void main(String[] args)
2    {
3    Data2 hoje = new Data2();
4    hoje.inicializaData(7,1,2001);
5    hoje.imprimeData();
6    }
```

Exercício 3.7: ★

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

```
1  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 3.8: ★

Escreva uma aplicação em Java que demonstre o uso de instâncias da classe `Contador`, que deve ter sido criada como resposta ao exercício 2.29.

Exercício 3.9: ★

Escreva uma aplicação em Java que demonstre o uso de instâncias da classe `PoligonoRegular`, que deve ter sido criada como resposta ao exercício 2.37.

Exercício 3.10: ★★

Uma instância da classe `Ponto2D` foi criada na linha 41 da figura 3.5. Os dados encapsulados nessa instância podem ser modificados? Por quê?

Exercício 3.11: ★★

Usando as classes `Veiculo` e `VeiculoTerrestre`, que devem ter sido criadas como resposta aos exercícios 2.15 e 2.16, escreva uma aplicação em Java que declare várias instâncias das duas classes. Em particular, tente descrever o mesmo veículo usando duas instâncias, uma da classe `Veiculo` e outra da classe `VeiculoTerrestre`.

Exercício 3.12: ★★

Escreva uma aplicação em Java que demonstre o uso de instâncias das classes `Livro`, `LivroLivraria` e `LivroBiblioteca` (veja os exercícios 2.22, 2.23 e 2.24).

Exercício 3.13: ★★

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 2.30).

Exercício 3.14: ★★

Escreva uma aplicação em Java que demonstre o uso de instâncias da classe `DataHora` (veja o exercício 2.33).

Exercício 3.15: ★★

Escreva uma aplicação que demonstre o uso de instâncias da classe `ContaBancariaSimplificada` que deve ter sido criada como resposta ao exercício 2.39. 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).

Exercício 3.16: ★★★

Demonstre o uso de instâncias da classe `RegistroAcademico`, que deve ter sido escrita com todas as modificações sugeridas nos exercícios do capítulo 3. Veja os exercícios 2.46, 2.47 e 2.45.

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  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 2.51. Demonstre o uso de todas as operações.

3.2 Exercícios complementares do Capítulo 3

Exercício 3.19: ★

O que acontecerá se tentamos imprimir uma instância de uma classe que não tem o método `toString`? Demonstre com um pequeno programa.

Exercício 3.20: ★

Escreva o método `toString` na classe `Data`, de forma que o mês seja mostrado por extenso em vez de numeral (veja o exercício 2.13).

Exercício 3.21: ★

Escreva o método `clona` na classe `Data`, que retorne uma nova instância que é uma cópia da própria data.

Exercício 3.22: ★

Escreva o método `toString` para as classes `Livro`, `LivroLivraria` e `LivroBiblioteca` (exercícios 2.22, 2.23 e 2.24).

Exercício 3.23: ★

Escreva o método `toString` para a classe `ContaBancariaSimplificada` (exercício 2.39).

Exercício 3.24: ★

Retire as partes que explicitam a conversão de dados nas linhas 35 e 42 da figura 3.1. O que acontece? Por quê?

Exercício 3.25: ★

Crie o método `criaRevertido` para a classe `Ponto2D` (figura 3.4) que retorne uma nova instância da classe onde os valores encapsulados `x` e `y` são revertidos.

Exercício 3.26: ★★

Escreva o método `toString` para a classe `Retangulo` (exercício 2.36), reaproveitando o método `toString` da classe `Ponto2D` (figura 3.4).

Exercício 3.27: ★★

Escreva o método `toString` para a classe `ModeloDeComputador` (exercício 2.43).

Exercício 3.28: ★★

Crie o método `temEixoComum` para a classe `Ponto2D` (figura 3.4) que receba uma outra instância da classe `Ponto2D` e retorne o valor booleano `true` se as coordenadas horizontais e/ou verticais encapsuladas forem iguais às da instância passada como argumento. Por exemplo, as coordenadas $(1,2)$ e $(1,-17)$ têm eixo comum, as $(-9,0)$ e $(-9,0)$ também têm, mas as $(13,-8)$ e $(8,-22)$ não têm eixo comum.

Exercício 3.29: ★★

Crie o método `distância` para a classe `Ponto2D` (figura 3.4) que recebe uma outra instância da classe `Ponto2D` e retorna um valor do tipo `double` correspondente à distância euclidiana entre o `Ponto2D` encapsulado e o passado como argumento. *Dica:* A distância euclidiana `d` entre um ponto com coordenadas (x_1,y_1) e outro ponto com coordenadas (x_2,y_2) é calculada por $\sqrt{(x_1-x_2)^2+(y_1-y_2)^2}$, que pode ser escrita em Java como `d = Math.sqrt((`x_1`-`x_2`)*(`x_1`-`x_2`)+(`y_1`-`y_2`)*(`y_1`-`y_2`))`, onde `Math.sqrt` é o método em Java que calcula a raiz quadrada.

Exercício 3.30: ★★

Crie o método `distânciaDaOrigem` para a classe `Ponto2D` (figura 3.4) que não recebe nenhum argumento, mas calcula a distância euclidiana entre as coordenadas encapsuladas e a origem do sistema de coordenadas. Para isso, dentro do método, crie uma instância de `Ponto2D` correspondente à origem e passe-a como argumento para o método `distância`, que deve ter sido escrito como resposta ao exercício 3.29.

Exercício 3.31: ★★★

Crie o método `éPróximo` para a classe `Ponto2D` (figura 3.4) que recebe uma outra instância da classe `Ponto2D` e um limiar (valor do tipo `double`) como argumentos, calculando a distância entre as coordenadas encapsuladas e as coordenadas da instância passada como argumento, retornando o valor booleano `true` se a distância for menor do que o limiar passado como argumento. Por exemplo, se o ponto encapsulado vale (3,3), o passado como argumento vale (4,1). Se o limiar for 3.0, o método deve retornar `true` já que a distância entre os dois pontos (2.236) é menor do que o limiar. Se o limiar fosse 2.0, o método deverá retornar `false`.

Capítulo 4

Construtores e sobrecarga

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 2.12). 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: ⋆

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

```
1  class Data
2    {
3    private byte dia,mês;
4    private short ano;
5    private Data(byte d,byte m,short a)
6      {
7      dia = d; mês = m; ano = a;
8      }
9    } // fim da classe
```

Exercício 4.5: ⋆

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

```
1  class Ponto2D
2    {
3    private double x,y;
4    Ponto2D()
5      {
6      Ponto2D(0.0,0.0);
7      }
8    Ponto2D(double coord1,double coord2)
9      {
10     x = coord1; y = coord2;
11     }
12   } // fim da classe
```

Exercício 4.6: ⋆

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

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

Exercício 4.7: ⋆

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

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

Exercício 4.8: ⋆

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

```
1  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   } // fim da classe
```

Exercício 4.9: ⋆

Liste as assinaturas dos construtores e métodos na classe `RoboSimples` (figura 4.7).

Exercício 4.10: ⋆

Escreva dois construtores para a classe `Contador`, 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 4.11: ⋆

Considerando a classe `RoboSimples` (figura 4.7), 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 4.12: ⋆

Escreva um construtor para a classe `PoligonoRegular` (exercício 2.37), que receba um valor inteiro correspondente ao número de lados do polígono.

Exercício 4.13: ⋆⋆

O que aconteceria com a instância `clonado` da classe `RoboSimples` (declarada na linha 29 da figura 4.8) se a declaração fosse `RoboSimples clonado = new RoboSimples(""+número5);`?

Exercício 4.14: ⋆⋆

Escreva dois construtores para a classe `ContaBancariaSimplificada` (exercício 2.39), 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 4.15: ⋆⋆

Explique, com suas palavras, o que acontecerá se sobrecarregarmos o método `toString`.

Exercício 4.16: ⋆⋆

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 4.17: ⋆⋆

Suponha que os robôs modelados pela classe `RoboSimples` 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 4.18: ★★

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

```
class Soma
  {
  public int doisValores(int a,int b) // soma dois inteiros
    {
    return a+b;
    }
  public double doisValores(double a,int b) // soma um double e um inteiro
    {
    return a+b;
    }
  public double doisValores(double a,double b) // soma dois doubles
    {
    return a+b;
    }
  }

class TesteSoma
  {
  public static void main(String[] args)
    {
    Soma soma = new Soma(); // cria instância da classe Soma
    // Declara várias variáveis
    byte b = 20;
    short s = 99;
    int i = 1000;
    long l = 1234L;
    float f = 3.1416f;
    double d = 2000;
    // Chama vários métodos da classe Soma
    System.out.println(soma.doisValores(b,s));
    System.out.println(soma.doisValores(i,s));
    System.out.println(soma.doisValores(i,i));
    System.out.println(soma.doisValores(l,b));
    System.out.println(soma.doisValores(f,s));
    System.out.println(soma.doisValores(d,b));
    System.out.println(soma.doisValores(b,d));
    System.out.println(soma.doisValores(i,l));
    System.out.println(soma.doisValores(l,l));
    System.out.println(soma.doisValores(d,f));
    }
  }
```

Exercício 4.19: ★★

Escreva dois construtores para a classe `Ponto2D` (figura 3.4): 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 4.20: ★★

Escreva três construtores para a classe `NumeroComplexo` (exercício 2.51). 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 4.21: ★★★

Escreva uma classe `SiteNaInternet` que represente os dados de um site na Internet, como seu título ou nome, sua URL, a data de primeiro acesso, a data de último acesso e o número de visitas. Use a classe `Contador` (exercício 2.29) para contar o número de visitas, e a classe `DataHora` (exercício 2.33) para representar as datas de primeiro e último acesso. Escreva, para essa classe, o construtor que inicializa todos os campos e o método `toString`.

Exercício 4.22: ★★★

Escreva quatro construtores para a classe `Linha` (exercício 2.35): 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 4.23: ★★★

Escreva quatro construtores para a classe `Retangulo` (exercício 2.36): 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.

4.2 Exercícios complementares do Capítulo 4

Exercício 4.24: ★

Se a classe `Data` tivesse um método `toString` (pedido no exercício 3.20), o método `toString` da classe `EventoAcademico` (figura 4.3) poderia ser escrito de maneira muito mais simples. Reescreva o método `toString` da classe `EventoAcademico` usando uma chamada ao método `toString` da classe `Data`.

Exercício 4.25: ★

Quando chamamos o método `inicializaData` para inicializar campos de uma instância da classe `Data`, somos obrigados a fazer o *cast* para que os argumentos para o método sejam reconhecidos como sendo dos tipos apropriados (ver linhas 35 e 42 da figura 3.1). Usando sobrecarga, seria possível escrever um outro método `inicializaData` que recebe três argumentos do tipo `int`, que serão convertidos dentro do corpo do método, fazendo com que o programador usuário da classe não precise se preocupar com o *cast* em seus programas. Embora a solução possa ser compilada e executada, existe um pequeno problema de *design* ou de organização de classes com essa solução. Qual é?

Exercício 4.26: ★

Modifique o método `inicializaRegistroAcademicoSemConstrutor` da classe `RegistroAcademicoSemConstrutor` para que o valor passado para o campo `percentualDeCobrança` esteja sempre correto (isto é, entre zero e cem). Caso o valor passado seja menor do que zero, deve ser considerado igual a zero; caso o valor seja maior do que cem, deve ser considerado como sendo igual a cem.

Exercício 4.27: ★★

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 30 e 33 da figura 4.3)? Explique, usando a aplicação na figura 4.4 como exemplo.

Exercício 4.28: ★★★

As coordenadas de posicionamento dos robôs modelados pela classe `RoboSimples` são representadas por valores separados (*x* e *y*). Reescreva a classe `RoboSimples` de forma que instâncias da classe `Ponto2D` sejam usadas para representar as coordenadas dos robôs. Para isso, reescreva também a própria classe `Ponto2D` para que seja possível modificar as coordenadas encapsuladas.

Capítulo 5

Campos e métodos estáticos

5.1 Exercícios do Capítulo 5

Exercício 5.1: ⋆

Explique, com suas palavras, por que os campos na classe `ConstantesMatematicas` (figura 5.5) não devem ser declarados com o modificador `private`.

Exercício 5.2: ⋆

Escreva, para a classe `DataComFabrica` (figura 5.11), um método `seteDeSetembro` que se comporte como uma fábrica de instâncias.

Exercício 5.3: ⋆

Escreva, para a classe `DataComFabrica` (figura 5.11), um método `primeiroDoMês` que se comporte como uma fábrica de instâncias.

Exercício 5.4: ⋆

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

Exercício 5.5: ⋆⋆

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

Exercício 5.6: ⋆⋆

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 5.7: ★★

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 5.8: ★★

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 5.9: ★★

Escreva uma classe `ConversaoDeTemperatura` 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 5.10: ★★

O que aconteceria se não inicializássemos os campos da listagem na figura 5.5? Explique.

Exercício 5.11: ★★★

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 5.12: ★★★

Escreva uma classe que contenha métodos estáticos para calcular as médias e somas de dois, três, quatro e cinco valores, considerando que os argumentos e retorno dos métodos podem ser dos tipos `int` e `double`. Um total de 16 métodos deverão ser criados.

Exercício 5.13: ★★★

Escreva uma versão da classe `RegistroAcademico` que tenha o campo `númeroDeMatrícula` 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. *Dica:* Use a figura 4.1 como base.

Exercício 5.14: ★★★

Escreva uma versão da classe `ContaBancariaSimplificada` que tenha 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. *Dica:* Use o modelo na figura 1.5 como base.

Exercício 5.15: ★★★

Escreva uma classe `SerieLimitada`, que encapsula um valor inteiro seqüencial 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 seqüencial. 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.

5.2 Exercícios complementares do Capítulo 5

Exercício 5.16: ★

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

Exercício 5.17: ★

A distância média da Terra à Lua é de aproximadamente 382.000 quilômetros. Usando a classe `ConversaoDeUnidades-DeComprimento` (figura 5.7), 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 5.18: ★

A área de um campo de futebol é de 8.250 metros quadrados. Usando a classe `ConversaoDeUnidadesDeArea` (exercício 5.6), 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 5.19: ★

O volume de uma piscina olímpica é de 1.890 metros cúbicos. Usando a classe `ConversaoDeUnidadesDeVolume` (exercício 5.7), 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 5.20: ★

O tempo de gestação de um elefante indiano é de aproximadamente 624 dias. Usando a classe `ConversaoDeUnidades-DeTempo` (exercício 5.8), 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 `ConversaoDeUnidadesDeVelocidade`, se necessário.

Exercício 5.21: ★

Escreva um programa em Java que, usando a classe `ConversaoDeTemperatura` (exercício 5.9), mostre quantos graus Kelvin e Fahrenheit correspondem a zero e cem graus Celsius.

Exercício 5.22: ⋆⋆

Existe um problema em potencial com a classe `SimuladorDeCaixaDeBanco` (figura 5.4): se uma nova instância da classe for criada em uma aplicação onde já existam algumas instâncias sendo usadas, o número do cliente será resetado (voltará a ser zero). Modifique a classe para prevenir esse problema.

Capítulo 6

Estruturas de decisão e controle – condicionais

6.1 Exercícios do Capítulo 6

Exercício 6.1: ⋆

Escreva para a classe `Comparavel` (figura 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` (figura 6.3) 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: ⋆

O método `mudaDireção` da classe `RoboSimples` (figura 4.7) 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 6.5: ⋆

Modifique o método `calculaPreço` da classe `EntradaDeCinema` (figura 6.3) para que este também considere que horas são, e retorne o preço de meia entrada para antes de quatro horas.

Exercício 6.6: ⋆

O que aconteceria se todos os `else` fossem retirados do método `mostraData` da classe `DataIf` (figura 6.4)? Existe alguma vantagem ou desvantagem em fazer isso?

Exercício 6.7: ⋆

Modifique o método `diasNoMês` da classe `DataSwitch` (figura 6.6) para que ele use comandos `if` em vez de `switch`.

Exercício 6.8: ⋆

Ao final do laço que controla as tentativas de acerto no método `tenta` da classe `JogoDeAdivinhacao` (figura 7.5, linhas 53 a 56), o número de tentativas é avaliado para verificar se o usuário acertou ou não o número secreto. Que outra condição poderia ser usada para essa verificação? Modifique o método `tenta` para usar essa outra condição.

Exercício 6.9: ⋆

Explique e exemplifique o que aconteceria com o método `diasNoMês` da classe `DataSwitch` (figura 6.6) se os comandos `break` fossem retirados das instruções `case` do método.

Exercício 6.10: ⋆

Modifique o método `calculaMensalidade` na classe `RegistroAcademicoSemConstrutor` (mostrado na figura 4.1) para que este use a instrução `switch` em vez de um bloco de `if-else`s.

Exercício 6.11: ⋆

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

```
1  class MultiplaEscolha
2    {
3    public static void main(String[] argumentos)
4      {
5      System.out.println("Escolha a opção correta:");
6      System.out.println("P - Quantos lados tem um círculo ?");
7      System.out.println("1 - Nenhum.");
8      System.out.println("2 - Dois.");
9      System.out.println("3 - Infinitos.");
10     System.out.println("4 - Nenhuma das respostas acima.");
11
12     byte resposta = Keyboard.readByte();
13     switch(resposta)
14       {
15       default   : System.out.println("Você não forneceu resposta !"); break;
16       case     4: System.out.println("Correto."); break;
17       case     1:
18       case     2:
19       case     3:
20       case     4: System.out.println("Não estou bem certo..."); break;
21       default   : System.out.println("Errado."); break;
22       }
23     }
24   } // fim da classe
```

Exercício 6.12: ⋆⋆

Escreva para a classe `Comparavel` (figura 6.1) 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 6.13: ⋆⋆

Escreva, para a classe `Ponto2D` (mostrada na figura 3.4), 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. Veja também a figura 3.3.

Exercício 6.14: ⋆⋆

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 que imprima o nome da carta por extenso, usando a instrução `switch`.

Exercício 6.15: ★★

Escreva versões do método `éDiferenteDe` (veja o exercício 6.12) 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 6.16: ★★★

Escreva um programa 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. O programa deve considerar divisões por zero como sendo erros, e imprimir uma mensagem adequada.

Exercício 6.17: ★★★

Modifique o programa do exercício 6.16 para que, caso o segundo valor entrado seja igual a zero, o programa não permita a escolha da operação divisão.

Exercício 6.18: ★★★★

Modifique a classe `Retangulo` (exercício 2.36) para que esta contenha um método `calculaIntersecção`, que recebe como argumento uma outra instância da própria classe `Retangulo` e calcule 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 retângulo-intersecção podem ser calculados com regras simples, implementadas através de `if`s encadeados. Nem sempre existe intersecção entre dois retângulos. Considere a figura 6.1: 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 2.50.

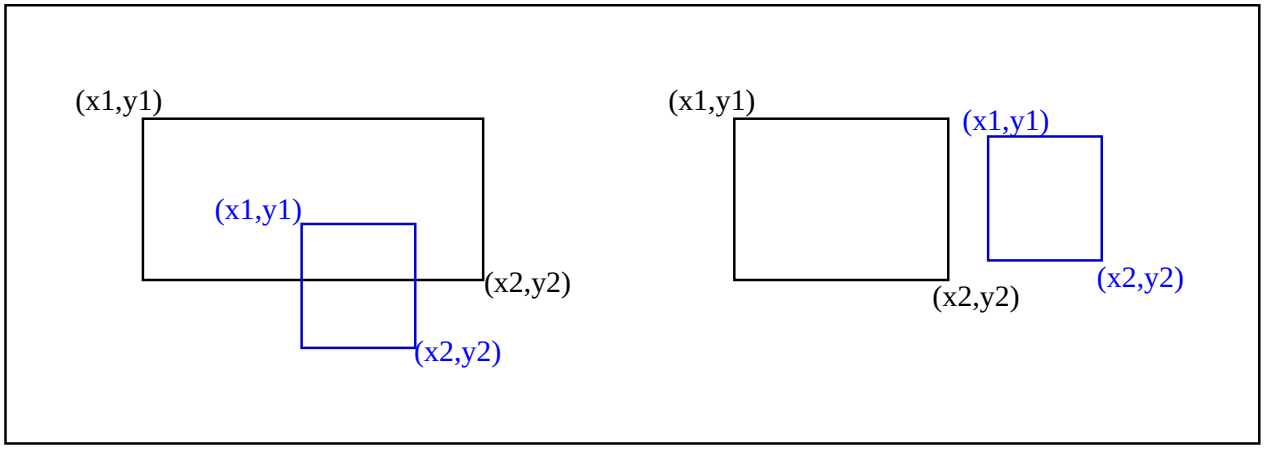


Figura 6.1: Intersecção de dois retângulos.

Exercício 6.19: ★★★★

Usando o exercício 6.18 como base, escreva mais dois métodos na classe `Retangulo` que recebam uma instância da classe `Retangulo` como argumento e retornem, separadamente, a área e o perímetro da intersecção do retângulo encapsulado e do retângulo passado como argumento.

6.2 Exercícios complementares do Capítulo 6

Exercício 6.20: ★★

Modifique a classe `ContaBancariaSimplificada` (exercício 2.39) para que o método `retira` permita a retirada de valores de contas especiais de forma que o saldo negativo máximo seja de 1000 reais, ou seja, não permitindo que contas especiais fiquem com débito maior do que 1000 reais.

Exercício 6.21: ★★

O método `calculaMensalidade` da classe `RegistroAcademicoSemConstrutor` tem um ponto ineficiente: mesmo que o valor do campo `percentualDeCobrança` seja igual a zero, vários comandos `if` serão executados para verificar qual deve ser a mensalidade dependendo do código do curso. Modifique o método para que este seja mais eficiente.

Exercício 6.22: ★★★

Escreva uma versão da classe `RegistroAcademico` (usando como base a classe `RegistroAcademicoSemConstrutor` na figura 4.1) para que esta tenha um construtor que receba o nome do aluno, um número de matrícula, um código do curso e um percentual de cobrança. O construtor deverá garantir que o percentual de cobrança esteja entre zero e cem por cento, assumindo que, se um valor inválido for passado, o valor será considerado como sendo igual a cem. O construtor também deverá verificar se o código do curso é um dos valores reconhecidos (que pode ser 34 para `"Ciência da Computação"`, 39 para `"Engenharia da Computação"`, 41 para `"Arquitetura"`, 43 para `"Engenharia Civil"` e 45 para `"Engenharia Elétrica"`). Qualquer valor diferente destes deverá ser considerado zero.

Exercício 6.23: ★★★

Escreva para a classe `RegistroAcademico` (exercício 6.22) um método `toString` que retorne o nome, número de matrícula e nome do curso do aluno encapsulado pela classe. Para retornar o nome do aluno considere a lista de cursos mostrada no exercício 6.22. Se o código do curso for zero (indicando inválido), considere o nome do curso como sendo `"Incorreto"`.

Capítulo 7

Estruturas de decisão e controle – repetição

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: ⋆

Durante a execução do seguinte trecho de código

```
for(System.out.print("Início,");true;System.out.print("Fim."))
    System.out.print("Executando,");
```

o que será mostrado no terminal?

A. `Início,Executando,Executando,Executando,...` (repetido infinitamente)
B. `Início,Executando,Fim.Executando,Fim.Executando,Fim...` (repetido infinitamente)
C. `Início,Executando,Fim.` (sem repetição)
D. Um erro de execução ocorrerá.

Exercício 7.5: ⋆

O que acontecerá se removermos a linha 42 da figura 7.4? Explique.

Exercício 7.6: ★

O que aconteceria se o valor inicial da variável `fatorial` no método `fatorial` da classe `ProbabilidadeBasica` (figura 7.8) fosse inicializado com zero?

Exercício 7.7: ★

Reescreva os laços `while` na classe `DemoWhile` (figura 7.1) usando laços `for`.

Exercício 7.8: ★

Reescreva os laços `for` na classe `DemoFor` (figura 7.7) usando laços `while` ou `do-while`.

Exercício 7.9: ★

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

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

Exercício 7.10: ★

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

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

Exercício 7.11: ★

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

```
1  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   } // fim da classe
```

Exercício 7.12: ★

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

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: ★★

Usando a classe `JogoDeAdivinhacao` (figura 7.5) 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 7.16: ★★

Reescreva o método `tenta` da classe `JogoDeAdivinhacao` (figura 7.5) 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 7.17: ★★

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 7.18: ★★

Usando o método `currentTimeMillis` da classe `System` (veja a seção 7.3), calcule quantos movimentos de anéis o programa que resolve o problema das Torres de Hanói (figura 7.13) calcula por segundo. Considerando que para mover sessenta e quatro anéis serão necessários $2^{64} - 1$ movimentos, que os monges tibetanos são tão rápidos quanto o computador usado nos cálculos, e que esses monges previram que o mundo terminaria quando os anéis fossem todos movidos do primeiro para o último pino, calcule aproximadamente o ano no qual o mundo terminará.

Exercício 7.19: ★★★

Modifique as classes `JogoDeAdivinhacao` e `DemoJogoDeAdivinhacao` (listagens nas figuras 7.5 e 7.6) 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 7.20: ★★★

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:

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

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

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

Exercício 7.21: ★★★

Modifique a classe `Serie` (exercício 7.20) 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).
Modifique também o construtor da classe `Serie` para que ele mostre uma mensagem de erro caso o valor passado para o multiplicador seja zero (para evitar que a série seja composta somente de zeros).

Exercício 7.22: ★★★

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 7.23: ★★★

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. O algoritmo é como segue:

- 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 pode ser calculado com o método `Math.abs`.

Exercício 7.24: ★★★

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 $A(3,4)$ gerou mais de 10.000 chamadas recursivas.

Exercício 7.25: ★★★★

Implemente uma solução recursiva para a geração de séries de números do exercício 7.20, 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?

7.2 Exercícios do Capítulo 7 que envolvem séries matemáticas

A criação de classes e programas que calculam séries matemáticas requer que o programador escreva laços com condicionais de diferentes graus de complexidade. Para reforçar os conceitos apresentados neste capítulo e no anterior, vários exercícios dessa natureza são apresentados.

Exercício 7.26: ★★

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 7.27: ★★

Escreva na classe `SeriesMatematicas` o método estático `piQuadradoSobre6` que calcule a série $\frac{1}{1^2}+\frac{1}{2^2}+\frac{1}{3^2}+\frac{1}{4^2}+\frac{1}{5^2}+\cdots$. 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. O resultado da série, se calculada infinitamente, será igual a $\frac{\pi^2}{6}$. Um recurso interessante é calcular o valor esperado da série com a expressão `(Math.PI * Math.PI)/6` e ver quão diferente desse valor esperado é o calculado pela série. A diferença pode ser mostrada passo a passo, para ver se a série está convergindo. Escreva também um programa que demonstre o método `piQuadradoSobre6`. Veja também o exercício 7.26.

Exercício 7.28: ★★

Escreva na classe `SeriesMatematicas` o método estático `piQuadradoSobre8` que calcule a série $\frac{1}{1^2}+\frac{1}{3^2}+\frac{1}{5^2}+\frac{1}{7^2}+\frac{1}{9^2}+\cdots$ com N termos, sendo que o valor de N deve ser fornecido como argumento ao método. O resultado da série, se calculada infinitamente, será igual a $\frac{\pi^2}{8}$. Veja também o exercício 7.26.

Exercício 7.29: ★★

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 $3/4$. Veja também o exercício 7.26.

Exercício 7.30: ★★

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 7.26.

Exercício 7.31: ★★

Escreva uma versão recursiva do método `elevadoA` (exercício 7.26) para a classe `SeriesMatematicas`.

Exercício 7.32: ★★★

Escreva um programa em Java que calcule a série $\frac{1}{1}-\frac{1}{2}+\frac{1}{3}-\frac{1}{4}+\frac{1}{5}-\frac{1}{6}+\cdots$ com N termos, sendo que o valor de N pode ser entrado via teclado. O resultado da série, se calculada infinitamente, será igual ao logaritmo com base natural de 2 ($ln(2) = 0.6931471805599453$). Que mecanismo poderá ser usado para efetuar a troca de sinais em cada passo da série? Veja também o exercício 7.26.

Exercício 7.33: ★★★

Escreva um programa em Java que calcule a série $\frac{1}{1}+\frac{1}{3}-\frac{1}{2}+\frac{1}{5}+\frac{1}{7}-\frac{1}{4}+\frac{1}{9}+\frac{1}{11}-\frac{1}{6}+\frac{1}{13}+\frac{1}{15}-\frac{1}{8}+\cdots$ com N termos, sendo que o valor de N pode ser entrado via teclado. O resultado da série, se calculada infinitamente, será igual ao logaritmo com base natural de 2 ($ln(2) = 0.6931471805599453$) multiplicado por $3/2$. Que mecanismo poderá ser usado para efetuar a troca de sinais em cada passo da série? Veja que a série se repete com grupos de dois termos positivos de denominador ímpar com um termo negativo de denominador par. *Dica:* Pode ser mais simples calcular vários termos da série em cada iteração do laço do que criar uma lógica que determine corretamente o sinal de cada termo. Veja também o exercício 7.26.

Exercício 7.34: ★★★

Escreva um programa em Java que calcule a série $\frac{1}{1}+\frac{1}{3}-\frac{1}{5}-\frac{1}{7}+\frac{1}{9}+\frac{1}{11}-\frac{1}{13}-\frac{1}{15}+\frac{1}{17}+\frac{1}{19}-\frac{1}{21}-\frac{1}{23}+\cdots$ com N termos, sendo que o valor de N pode ser entrado via teclado. O resultado da série, se calculada infinitamente, será igual a $\frac{\pi}{4}\sqrt{2}$. Como fazer a troca de sinais a cada dois passos da série? Veja que a série é composta de grupos de duas somas e duas subtrações. *Dica:* Pode ser mais simples calcular vários termos da série em cada iteração do laço do que criar uma lógica que determine corretamente o sinal de cada termo. Veja também os exercícios 7.26 e 7.33.

Exercício 7.35: ★★★

Escreva um programa em Java que calcule a série $1-\frac{1}{4}-\frac{1}{7}+\frac{1}{10}-\frac{1}{13}+\frac{1}{16}-\frac{1}{19}+\cdots$ com N termos, sendo que o valor de N pode ser entrado via teclado. O resultado da série, se calculada infinitamente, será igual a $\frac{1}{3}(\frac{\pi}{\sqrt{3}}+ln(2))$.

Exercício 7.36: ★★★

Escreva um programa em Java que calcule a série $1-\frac{1}{2}+\frac{1}{4}-\frac{1}{5}+\frac{1}{7}-\frac{1}{8}+\frac{1}{10}-\frac{1}{11}+\frac{1}{13}-\cdots$ com N termos, sendo que o valor de N pode ser entrado via teclado. O resultado da série, se calculada infinitamente, será igual a $\frac{\pi\sqrt{3}}{9}$. Aparentemente, não existe uma série para os denominadores dos termos, mas estes podem ser calculados de dois em dois facilmente. Veja também o exercício 7.33.

Exercício 7.37: ★★★

Escreva um programa em Java que calcule a série $(\frac{1}{1}+\frac{1}{2}-\frac{1}{3}+\frac{1}{4}-\frac{1}{5}-\frac{1}{6})+(\frac{1}{8}+\frac{1}{9}-\frac{1}{10}+\frac{1}{11}-\frac{1}{12}-\frac{1}{13})+(\frac{1}{15}+\frac{1}{16}-\frac{1}{17}+\frac{1}{18}-\frac{1}{19}-\frac{1}{20})+\cdots$ com N termos, sendo que o valor de N pode ser entrado via teclado. O resultado da série, se calculada infinitamente, será igual a $\frac{\pi\sqrt{7}}{7}$. Aparentemente, não existe uma série para os denominadores dos termos, mas estes podem ser calculados de seis em seis facilmente. Veja também o exercício 7.33.

Exercício 7.38: ★★★

Escreva na classe `SeriesMatematicas` o método estático `calculaPi` que calcule a série $2\times\frac{2}{1}\times\frac{2}{3}\times\frac{4}{3}\times\frac{4}{5}\times\frac{6}{5}\times\frac{6}{7}\times\cdots$ com N termos, sendo que o valor de N deve ser fornecido como argumento ao método. O resultado da série, se calculada infinitamente, será igual a π. Veja também o exercício 7.26.

Exercício 7.39: ★★★

Escreva na classe `SeriesMatematicas` o método estático `exponencial` que calcule a série $1+x+\frac{x^2}{2!}+\frac{x^3}{3!}+\frac{x^4}{4!}+\frac{x^5}{5!}+\cdots$ com N termos, sendo que os valores de N e x devem ser fornecidos como argumentos ao método. O resultado da série, se calculada infinitamente, será igual ao valor e^x (onde e é a base dos logaritmos naturais). *Dica:* O fatorial de um número pode ser calculado usando o método `fatorial` na classe `ProbabilidadeBasica` (figura 7.8). Veja também o exercício 7.26.

Exercício 7.40: ★★★

Escreva na classe `SeriesMatematicas` o método estático `seno` que calcule a série $x\times(1-\frac{x^2}{\pi^2})\times(1-\frac{x^2}{4\pi^2})\times(1-\frac{x^2}{9\pi^2})\times(1-\frac{x^2}{16\pi^2})\times\cdots$ com N termos, sendo que os valores de N e x devem ser fornecidos como argumentos ao método. O resultado da série, se calculada infinitamente, será igual ao seno do ângulo x em radianos. Use a constante `Math.PI` ou o método `calculaPi` (exercício 7.38) para obter o valor de π.

Exercício 7.41: ★★★

Escreva na classe `SeriesMatematicas` o método estático `senoDeOutraForma` que calcule a série $x-\frac{x^3}{3!}+\frac{x^5}{5!}-\frac{x^7}{7!}+\frac{x^9}{9!}-\cdots$ com N termos, sendo que os valores de N e x devem ser fornecidos como argumentos ao método. O resultado da série, se calculada infinitamente, será igual ao seno do ângulo x em radianos. O fatorial de um número pode ser calculado usando o método `fatorial` na classe `ProbabilidadeBasica` (figura 7.8), e o valor de x^y com o método `elevadoA` (exercício 7.26).

Exercício 7.42: ★★★

Escreva na classe `SeriesMatematicas` o método estático `senoDeXSobreX` que calcule a série $cos(\frac{x}{2})\times cos(\frac{x}{4})\times cos(\frac{x}{8})\times cos(\frac{x}{16})\times cos(\frac{x}{32})\times\cdots$ com N termos, sendo que os valores de N e x devem ser fornecidos como argumentos ao método. O resultado da série, se calculada infinitamente, será igual a $\frac{sin(x)}{x}$. O co-seno de um número pode ser calculado usando o método `Math.cos` que recebe o ângulo em radianos como argumento e retorna o co-seno desse ângulo.

Exercício 7.43: ★★★

Escreva um programa em Java que demonstre que a série $\frac{1}{1^2\times3^2}+\frac{1}{3^2\times5^2}+\frac{1}{5^2\times7^2}+\frac{1}{7^2\times9^2}+\frac{1}{9^2\times11^2}+\cdots$ é aproximadamente igual a $\frac{\pi^2-8}{16}$. Use para isso a constante `Math.PI` ou o método `calculaPi` (exercício 7.38). Permita ao usuário entrar o número N de termos da série pelo teclado, para poder experimentar com vários valores.

Exercício 7.44: ★★★

Escreva um programa em Java que demonstre que a série $\frac{1}{1^2\times2^2\times3^2}+\frac{1}{2^2\times3^2\times4^2}+\frac{1}{3^2\times4^2\times5^2}+\frac{1}{4^2\times5^2\times6^2}+\cdots$ é aproximadamente igual a $\frac{4\pi^2-39}{16}$. Use para isso a constante `Math.PI` ou o método `calculaPi` (exercício 7.38). Permita ao usuário entrar o número N de termos da série pelo teclado, para poder experimentar com vários valores.

Exercício 7.45: ★★★

Escreva um programa em Java que demonstre que a série $1-\frac{1}{4}+\frac{1}{7}-\frac{1}{10}+\frac{1}{13}-\frac{1}{16}+\cdots$ é aproximadamente igual a $\frac{\pi\sqrt{3}}{9}+\frac{1}{3}ln(2)$. Use para isso a constante `Math.PI` ou o método `calculaPi` (exercício 7.38), a constante `raizDe3` da classe `ConstantesMatematicas` (figura 5.5) e o resultado do exercício 7.32. Permita ao usuário entrar o número N de termos da série pelo teclado, para poder experimentar com vários valores.

Exercício 7.46: ★★★

Escreva um programa em Java que verifique a igualdade $\sum_{i=0}^{\infty}a^i=\frac{1}{1-a}$ para qualquer valor real a maior que zero e menor que um. *Dica:* O método `elevadoA` (exercício 7.26) pode ser útil na solução desse problema.

Exercício 7.47: ★★★

Escreva uma versão recursiva do programa que calcula a seqüência que converge para $1/2$ (exercício 7.30).

Exercício 7.48: ★★★

Escreva uma versão recursiva do programa que calcula a seqüência que converge para $3/4$ (exercício 7.29).

Exercício 7.49: ★★★★

Escreva na classe `SeriesMatematicas` o método estático `arcoSeno` que calcule a série $x+\frac{1}{2}\frac{x^3}{3}+\frac{1\times3}{2\times4}\frac{x^5}{5}+\frac{1\times3\times5}{2\times4\times6}\frac{x^7}{7}+\frac{1\times3\times5\times7}{2\times4\times6\times8}\frac{x^9}{9}+\cdots$ com N termos, sendo que os valores de N e x devem ser fornecidos como argumentos ao método. O resultado da série, se calculada infinitamente, será igual ao arco seno do ângulo x em radianos, para $|x|<1$. O valor de x^y pode ser calculado com o método `elevadoA` (exercício 7.26). *Dica:* Escreva dois métodos auxiliares `multiplicaSériePar` e `multiplicaSérieImpar` que possam calcular $2\times4\times6\times8...N$ e $1\times3\times5\times7...N$.

Exercício 7.50: ★★★★

Escreva na classe `SeriesMatematicas` o método estático `calculaPiComMétodoDeEuler` que calcule a série $2\times\left(1+\frac{1}{1\times3}+\frac{1\times2}{1\times3\times5}+\frac{1\times2\times3}{1\times3\times5\times7}+\frac{1\times2\times3\times4}{1\times3\times5\times7\times9}+\frac{1\times2\times3\times4\times5}{1\times3\times5\times7\times9\times11}+\cdots\right)$ com N termos, sendo que o valor de N deve ser fornecido como argumento ao método. O resultado da série, se calculada infinitamente, será igual a π. Veja também o exercício 7.38. *Dica:* Para valores muito grandes de N é possível que o denominador seja igual a $Infinity$, fazendo com que a somatória passe a valer NaN. Tente encontrar o maior valor de N que dá um resultado diferente de NaN.

Exercício 7.51: ★★★★

Escreva na classe `SeriesMatematicas` o método estático `arcoTangente` que calcule uma das três séries

$$\begin{cases} -\frac{\pi}{2} - \frac{1}{x} + \frac{1}{3x^3} - \frac{1}{5x^5} + \frac{1}{7x^7} - \cdots & \text{se } x \leq 1 \\ x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \frac{x^9}{9} - \cdots & \text{se } -1 < x < 1 \\ +\frac{\pi}{2} - \frac{1}{x} + \frac{1}{3x^3} - \frac{1}{5x^5} + \frac{1}{7x^7} - \cdots & \text{se } x \geq 1 \end{cases}$$

com N termos, sendo que os valores de N e x devem ser fornecidos como argumentos ao método. A série a ser calculada depende do valor de x. O resultado da série, se calculada infinitamente, será igual ao arcotangente do ângulo x em radianos. *Dicas:* Veja que a diferença entre duas das séries é somente um sinal, e tente fazer sub-rotinas estáticas privadas que sejam usadas por esse método.

Exercício 7.52: ★★★★

Escreva uma versão recursiva do programa que calcula a seqüência que converge para $ln(2)$ (exercício 7.32) de forma recursiva.

Exercício 7.53: ★★★★

Escreva uma versão recursiva do método estático `senoDeXSobreX` (exercício 7.42) na classe `SeriesMatematicas`.

Exercício 7.54: ★★★★

Escreva uma versão otimizada do método `senoDeOutraForma` da classe `SeriesMatematicas` (veja o exercício 7.41). Escreva um programa que mostre que a versão otimizada é realmente mais rápida que a não-otimizada.

Exercício 7.55: ★★★★★

O valor $\frac{2}{\pi}$ pode ser calculado com a seguinte série infinita:

$$\frac{2}{\pi} = \frac{\sqrt{2}}{2} \frac{\sqrt{2+\sqrt{2}}}{2} \frac{\sqrt{2+\sqrt{2+\sqrt{2}}}}{2} \frac{\sqrt{2+\sqrt{2+\sqrt{2+\sqrt{2}}}}}{2} \frac{\sqrt{2+\sqrt{2+\sqrt{2+\sqrt{2+\sqrt{2}}}}}}{2} \cdots$$

Escreva um programa em Java que calcule o valor $\frac{2}{\pi}$ usando a série acima, com N termos, sendo que o valor de N pode ser lido do teclado. *Dica:* Um valor temporário calculado para um termo da série pode ser reutilizado para o cálculo do próximo valor.

Exercício 7.56: ★★★★★

O valor de $1/\pi$ pode ser calculado por qualquer uma das séries ou seqüências abaixo:

$$\frac{1}{\pi} = \sum_{n=0}^{n=\infty} \frac{5+42n}{2^{12n+4}} \left(\frac{(2n)!}{(n!)^2}\right)^3$$

$$\frac{1}{\pi} = \frac{1}{72} \sum_{n=0}^{n=\infty} (-1^n) \frac{(4n)!}{(n!)^4 4^{4n}} \frac{23+260n}{18^{2n}}$$

$$\frac{1}{\pi} = \frac{2\sqrt{2}}{9801} \sum_{n=0}^{n=\infty} \frac{(4n)!}{(n!)^4 4^{4n}} \frac{1103+26390n}{99^{4n}}$$

$$\frac{1}{\pi} = 12 \times \sum_{n=0}^{n=\infty} (-1^n) \frac{(6n)!}{(n!)^3 \times (3n)!} \frac{13591409+545140134n}{640320^{(3n+\frac{3}{2})}}$$

Escreva um programa em Java que, usando as séries acima, calcule o valor de $1/\pi$. Tente avaliar qual dos métodos é o mais eficiente: sabendo que $1/\pi$ é aproximadamente igual a 0.3183098861837907, execute os métodos até que a diferença entre o valor calculado e o valor constante seja muito pequena, e meça o tempo que cada um dos métodos demorou para atingir essa marca.

Exercício 7.57: ★★★★★

O valor de π também pode ser calculado indiretamente por qualquer uma das séries ou seqüências abaixo:

$$\pi\sqrt{2} = \sum_{n=0}^{n=\infty} \left(\frac{4}{6n+1} + \frac{1}{6n+3} + \frac{1}{6n+5}\right) \frac{(-1)^n}{8^n}$$

$$\frac{\pi^2}{18} = \sum_{n=0}^{n=\infty} \left(\frac{1}{(6n+1)^2} - \frac{3}{2(6n+2)^2} - \frac{1}{2(6n+3)^2} - \frac{3}{8(6n+4)^2} + \frac{1}{16(6n+5)^2}\right) \left(\frac{1}{64}\right)^n$$

$$\frac{8\pi^2}{9} = \sum_{n=0}^{n=\infty} \left(\frac{16}{(6n+1)^2} - \frac{24}{(6n+2)^2} - \frac{8}{(6n+3)^2} - \frac{6}{(6n+4)^2} + \frac{1}{(6n+5)^2}\right) \left(\frac{1}{64}\right)^n$$

Escreva um programa em Java que, usando as séries acima, calcule o valor de π. Tente avaliar qual dos métodos é o mais eficiente: sabendo que π é aproximadamente igual a 3.141592653589793, execute os métodos até que a diferença entre o valor calculado e o valor constante seja muito pequena, e meça o tempo que cada um dos métodos demorou para atingir essa marca. Tente incluir no cálculo do tempo gasto o tempo necessário para converter o valor da série ou somatória para π.

Exercício 7.58: ★★★★★

O logaritmo natural de 2 ($\ln(2)$) pode ser calculado por qualquer uma das séries ou seqüências abaixo:

$$\ln(2) = \frac{2}{3} \sum_{n=0}^{n=\infty} \frac{1}{(2n+1)9^n}$$

$$\ln(2) = \frac{1327}{1920} + \frac{45}{4} \sum_{n=4}^{n=\infty} \frac{(-1)^n}{n(n^2-1)(n^2-4)(n^2-9)}$$

$$\ln(2) = \frac{2}{3} + \sum_{n=1}^{n=\infty} \left(\frac{1}{2n} + \frac{1}{2n+1} + \frac{1}{8n+4} + \frac{1}{16n+12}\right) \frac{1}{16^n}$$

Escreva um programa em Java que, usando as séries acima, calcule o valor de $\ln(2)$. Tente avaliar qual dos métodos é o mais eficiente: sabendo que $\ln(2)$ é aproximadamente igual a 0.6931471805599453, execute os métodos até que a diferença entre o valor calculado e o valor constante seja muito pequena, e meça o tempo que cada um dos métodos demorou para atingir essa marca.

Exercício 7.59: ★★★★★

O valor de π também pode ser calculado por qualquer uma das séries ou seqüências abaixo:

$$\pi = \sum_{n=0}^{n=\infty} \left(\frac{2}{4n+1} + \frac{2}{4n+2} + \frac{1}{4n+3} \right) \frac{(-1)^n}{4^n}$$

$$\pi = \sum_{n=0}^{n=\infty} \left(\frac{4}{8n+1} - \frac{2}{8n+4} - \frac{1}{8n+5} - \frac{1}{8n+6} \right) \frac{1}{16^n}$$

$$\pi = \sum_{n=0}^{n=\infty} \left(\frac{2}{8n+1} + \frac{2}{8n+2} + \frac{1}{8n+3} - \frac{1}{16n+10} - \frac{1}{16n+12} - \frac{1}{32n+28} \right) \frac{1}{16^n}$$

Escreva um programa em Java que, usando as séries acima, calcule o valor de π. Tente avaliar qual dos métodos é o mais eficiente: sabendo que π é aproximadamente igual a 3.141592653589793, execute os métodos até que a diferença entre o valor calculado e o valor constante seja muito pequena, e meça o tempo que cada um dos métodos demorou para atingir essa marca.

Exercício 7.60: ★★★★★

O valor de π também pode ser calculado com qualquer uma das chamadas *fórmulas de Machin* mostradas abaixo:

$$\frac{\pi}{2} = 2 \arctan\left(\frac{1}{\sqrt{2}}\right) - \arctan\left(\frac{1}{\sqrt{8}}\right)$$

$$\frac{\pi}{4} = \arctan\left(\frac{1}{2}\right) + \arctan\left(\frac{1}{3}\right)$$

$$\frac{\pi}{4} = 2 \arctan\left(\frac{1}{2}\right) - \arctan\left(\frac{1}{7}\right)$$

$$\frac{\pi}{4} = 2 \arctan\left(\frac{1}{3}\right) + \arctan\left(\frac{1}{7}\right)$$

$$\frac{\pi}{4} = 4 \arctan\left(\frac{1}{5}\right) - \arctan\left(\frac{1}{239}\right)$$

Escreva um programa em Java que, usando as séries acima, calcule o valor de π, usando o método `arcoTangente` da classe `SeriesMatematicas` (exercício 7.51). Tente avaliar qual dos métodos é o mais eficiente: sabendo que π é aproximadamente igual a 3.141592653589793, execute os métodos até que a diferença entre o valor calculado e o valor constante seja muito pequena, e meça o tempo que cada um dos métodos demorou para atingir essa marca.

Exercício 7.61: ★★★★★

A raiz quadrada de 2 e seu inverso podem ser calculados com qualquer uma das seguintes séries:

$$\sqrt{2} = 1 + \sum_{n=1}^{n=\infty} (-1)^{n-1} \frac{(2n-2)!}{n!(n-1)!2^{2n-1}}$$

$$\sqrt{2} = \prod_{n=1}^{n=\infty} \left(1 + \frac{(-1)^{n-1}}{2n-1}\right)$$

$$\frac{1}{\sqrt{2}} = 1 + \sum_{n=1}^{n=\infty} (-1)^{n} \frac{(2n-1)!}{n!(n-1)!2^{2n-1}}$$

$$\frac{1}{\sqrt{2}} = \prod_{n=1}^{n=\infty} \left(1 - \frac{1}{4(2n-1)^2}\right)$$

Onde $\prod$ indica que os termos da série devem ser multiplicados. Escreva um programa em Java que, usando as séries acima, calcule o valor de $\sqrt{2}$. Tente avaliar qual dos métodos é o mais eficiente: sabendo que $\sqrt{2}$ é aproximadamente igual a 1.4142135623730950488, execute os métodos até que a diferença entre o valor calculado e o valor constante seja muito pequena, e meça o tempo que cada um dos métodos demorou para atingir essa marca.

Exercício 7.62: ★★★★★

A constante de Euler (γ) pode ser calculada por qualquer uma das séries ou seqüências abaixo:

$$\begin{aligned}\gamma &= \frac{1}{2}\times\left(\frac{1}{1^2}+\frac{1}{2^2}+\frac{1}{3^2}+\frac{1}{4^2}+\cdots\right)-\frac{1}{3}\times\left(\frac{1}{1^3}+\frac{1}{2^3}+\frac{1}{3^3}+\frac{1}{4^3}+\cdots\right)+\\ &\quad\frac{1}{4}\times\left(\frac{1}{1^4}+\frac{1}{2^4}+\frac{1}{3^4}+\frac{1}{4^4}+\cdots\right)-\frac{1}{5}\times\left(\frac{1}{1^5}+\frac{1}{2^5}+\frac{1}{3^5}+\frac{1}{4^5}+\cdots\right)+\cdots\end{aligned}$$

$$\begin{aligned}\gamma &= \frac{1}{2}\times\left(\frac{1}{2^2}+\frac{1}{3^2}+\frac{1}{4^2}+\frac{1}{5^2}+\cdots\right)+\frac{2}{3}\times\left(\frac{1}{2^3}+\frac{1}{3^3}+\frac{1}{4^3}+\frac{1}{5^3}+\cdots\right)+\\ &\quad\frac{3}{4}\times\left(\frac{1}{2^4}+\frac{1}{3^4}+\frac{1}{4^4}+\frac{1}{5^4}+\cdots\right)+\frac{4}{5}\times\left(\frac{1}{2^5}+\frac{1}{3^5}+\frac{1}{4^5}+\frac{1}{5^5}+\cdots\right)+\cdots\end{aligned}$$

Escreva um programa em Java que, usando as séries acima, calcule o valor de γ. Tente avaliar qual dos métodos é o mais eficiente: sabendo que γ é aproximadamente igual a 0.5772156649, execute os métodos até que a diferença entre o valor calculado e o valor constante seja muito pequena, e meça o tempo que cada um dos métodos demorou para atingir essa marca.

Exercício 7.63: ★★★★★

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

$$e = \frac{2}{1}\left(\frac{4}{3}\right)^{\frac{1}{2}}\left(\frac{6\times 8}{5\times 7}\right)^{\frac{1}{4}}\left(\frac{10\times 12\times 14\times 16}{9\times 11\times 13\times 15}\right)^{\frac{1}{8}}\left(\frac{18\times 20\times 22\times 24\times 26\times 28\times 30\times 32}{17\times 19\times 21\times 23\times 25\times 27\times 29\times 31}\right)^{\frac{1}{16}}\cdots$$

$$e = \left(\frac{2}{1}\right)^{\frac{1}{2}}\left(\frac{2\times 4}{3\times 3}\right)^{\frac{1}{4}}\left(\frac{4\times 6\times 6\times 8}{5\times 5\times 7\times 7}\right)^{\frac{1}{8}}\left(\frac{8\times 10\times 10\times 12\times 12\times 14\times 14\times 16}{9\times 9\times 11\times 11\times 13\times 13\times 15\times 15}\right)^{\frac{1}{16}}\cdots$$

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 meça o tempo que cada um dos métodos demorou para atingir essa marca.

7.3 Exercícios complementares do Capítulo 7

Exercício 7.64: ★

Escreva um programa em Java que, usando a classe `ConversaoDeUnidadesDeComprimento` (figura 5.7), imprima as distâncias em milhas e pés correspondentes às distâncias em metros de zero a cem metros. Veja também o exercício 5.17.

Exercício 7.65: ★

Escreva um programa em Java que, usando a classe `ConversaoDeUnidadesDeTempo` (exercício 5.8), imprima o tempo em segundos, horas, semanas e anos correspondentes ao tempo em dias, de um a trinta e um dias. Veja também o exercício 5.20.

Exercício 7.66: ★

Escreva um programa em Java que, usando a classe `ConversaoDeTemperatura` (exercício 5.9), imprima as temperaturas em graus Kelvin e Fahrenheit correspondentes aos graus Celsius entre zero e cem graus. Veja também o exercício 5.21.

Exercício 7.67: ⋆

Escreva uma aplicação em Java que crie e use instâncias das classes `EscolhaComWhile` (figura 7.2) ou `EscolhaComDoWhile` (figura 7.4).

Exercício 7.68: ⋆

Modifique as classes `JogoDeAdivinhacao` e `DemoJogoDeAdivinhacao` (listagens nas figuras 7.5 e 7.6) para que o número de tentativas seja também variável, isto é, que não seja fixo na classe `JogoDeAdivinhacao`.

Exercício 7.69: ⋆

O resultado da execução da classe `SomatoriaRecursiva` (figura 7.10) mostra erros de concordância quando o valor de N é igual a 1 ou zero – esses erros ocorreram porque o método `somatória` tinha que ser mantido simples para as explicações necessárias. Corrija o método `somatória` para que as mensagens sejam impressas corretamente.

Exercício 7.70: ⋆

Explique por que o método `somatória` na classe `SomatoriaRecursiva` (figura 7.10) retorna um valor do tipo `long` em vez de `int`.

Exercício 7.71: ⋆⋆

Um problema potencial das classes `EscolhaComWhile` (figura 7.2) e `EscolhaComDoWhile` (figura 7.2) é que seus construtores não verificam se o valor inicial é menor do que o final – se uma instância for construída com os valores `(1000,-1000)`, o método `escolhe` nunca terminará!
Reescreva os construtores de forma que, se o valor inicial da faixa for maior do que o final, o construtor trocará os valores, fazendo com que o final vire o inicial e vice-versa.

Exercício 7.72: ⋆⋆

Reescreva a classe `EscolhaComWhile` (figura 7.2) para que o método `escolha` seja estático, recebendo os valores inicial e final da faixa de valores que podem ser escolhidos como argumentos. O que deverá ser feito do construtor da classe?

Exercício 7.73: ⋆⋆

Escreva um programa em Java que verifique a diferença de tempo de execução de um contador de um a um milhão usando laços `while`, `do-while` e `for`. Existem diferenças? Use também um laço infinito (cuja condição de avaliação seja `true`) que seja terminado por um `if` associado a um `break` (veja figura 7.7, linhas 41 a 46) para as comparações. Explique as diferenças, se estas existirem e forem significantes.

Exercício 7.74: ⋆⋆

Escreva um programa em Java que verifique a diferença de tempo de execução do cálculo do trigésimo elemento da série de Fibonacci, usando o algoritmo recursivo (figura 7.14) e o algoritmo não-recursivo (exercício 7.17). **Evite tentar calcular o tempo de execução do algoritmo recursivo de Fibonacci para valores muito grandes.**

Capítulo 8

Reutilização de classes

8.1 Exercícios do Capítulo 8

Exercício 8.1: ⋆

Seria 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 campos, e a classe `Aluno` que estende a classe `Pessoa` poderia ter o método `imprimeAluno` para imprimir seus campos, e outras classes teriam seus métodos específicos para imprimir seus dados. 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  class Ponto2D
2    {
3    private double x,y;
4    Ponto2D(double _x,double _y)
5      {
6      x = _x; y = _y;
7      }
8    }
9
10 class Ponto3D extends Ponto2D
11   {
12   private double z;
13   Ponto3D(double _x,double _y,double _z)
14     {
15     x = _x;
16     y = _y;
17     z = _z;
18     }
19   }
```

Exercício 8.3: ⋆

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

```
1  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    public String toString()
9      {
10     return super.toString()+" "+super.toString();
11     }
12   }
```

Exercício 8.4: ★

Explique com suas palavras o que aconteceria se removêssemos a palavra-chave `super` da linha 32 da classe `Funcionario` (figura 8.8).

Exercício 8.5: ★

Liste todos os campos da classe `ChefeDeDepartamento` (figura 8.9), inclusive os herdados, e indique quais podem ser modificados diretamente a partir da classe `ChefeDeDepartamento`.

Exercício 8.6: ★

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

Exercício 8.7: ★

O que aconteceria se declarássemos o método `quantasPrestações` da classe `Automovel` (figura 8.11) como sendo `final`? Explique.

Exercício 8.8: ★

Para cada uma das cinco instâncias das classes criadas no método `main` da classe `EmprestimoBancario` (figura 8.16), verifique se as expressões `instanceof Pessoa`, `instanceof Funcionario` e `instanceof ChefeDeDepartamento` retornariam verdadeiro ou falso.

Exercício 8.9: ★★

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

Exercício 8.10: ★★

Escreva, para a classe `Funcionario0` (figura 8.5), um método `modificaNome` que permita a modificação do nome do funcionário. Considere as regras de modificação de acesso dos campos das classes envolvidas.

Exercício 8.11: ★★

Considere a classe `DataHora` (figura 8.1) 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`. Veja também a figura 2.4.

Exercício 8.12: ★★

Escreva outro construtor para a classe `ChefeDeDepartamento` (figura 8.9) 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. Use o construtor da classe `DataHora` (figura 8.1) como exemplo. Veja também o exercício 8.15.

Exercício 8.13: ★★

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

Exercício 8.14: ★★

Escreva a classe `PacienteDeClinica`, descrita na figura 8.10.

Exercício 8.15: ★★

Escreva outro construtor para a classe `Funcionario` (figura 8.8) que em vez de receber e repassar os dados separados de uma pessoa (`nome` e `identidade`), receba uma instância da classe `Pessoa` já construída. Use o construtor da classe `DataHora` (figura 8.1) como exemplo.

Exercício 8.16: ★★

Escreva duas versões da classe `DataHora`: 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 funcionais entre as classes?

Exercício 8.17: ★★

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

```
1  class Data
2    {
3    private byte dia,mês; private short ano;
4    Data(byte d,byte m,short a)
5      {
6      dia = d; mês = m; ano = a;
7      }
8    public String toString()
9      {
10     return dia+"/"+mês+"/"+ano;
11     }
12   }
13
14 class DataHora extends Data
15   {
16   private Hora hora;
17   public DataHora(byte d,byte m,short a,byte hor,byte min,byte seg)
18     {
19     super(d,m,a);
20     hora = new Hora(hor,min,seg);
21     }
22   public String toString()
23     {
24     return super.toString()+" "+hora.toString();
25     }
26   }
27
28 class EntradaNaAgenda extends DataHora
29   {
30   private String evento;
31   public EntradaNaAgenda(byte d,byte m,short a,byte hor,byte min,byte seg,String ev)
32     {
33     super.super(d,m,a);
34     super(d,m,a,hor,min,seg);
35     evento = ev;
36     }
37   public String toString()
38     {
39     return super.super.toString()+":"+super.toString()+" -> "+ev;
40     }
41   }
```

Exercício 8.18: ★★

Escreva as classes `LampadaTresEstados` e `Lampada100Estados`, que herdam da classe `Lampada`. Veja também os exercícios 2.18 e 2.19. Que métodos devem ser superpostos?

Exercício 8.19: ★★

Troque a ordem de verificação de instâncias na segunda versão do método `calculaEmpréstimo` da classe `EmprestimoBancario` (figura 8.16). Compile e execute o programa, e explique o resultado da modificação.

Exercício 8.20: ★★

Reescreva as classes `Automovel` e `AutomovelDeLuxo` (listagens nas figuras 8.11 e 8.13) para que o campo `NÚMEROMÁXIMODEPRESTAÇÕES` não seja mais usado, eliminando a necessidade de ocultá-lo. *Dica:* O valor do número máximo de prestações pode ser retornado como uma constante pelo método `quantasPrestações`, que pode ser superposto nas classes descendentes se o valor do número máximo de prestações for diferente das classes ancestrais. Alguma modificação deve ser feita na classe `AutomovelBasico` (figura 8.12), que herda da classe `Automovel` e é a superclasse de `AutomovelDeLuxo`? Alguma modificação deve ser feita na classe `DemoAutomoveis` (figura 8.14), que demonstra os usos das classes?

Exercício 8.21: ★★

Modifique os métodos `calculaEmpréstimo` da classe `EmprestimoBancario` (figura 8.16) para que o empréstimo calculado para uma instância que herde da classe `Aluno` seja constante e igual a 500 reais.

Exercício 8.22: ★★

O que acontecerá se trocarmos a linha `Funcionario temporário = (Funcionario)p;` por `ChefeDeDepartamento temporário = (ChefeDeDepartamento)p;` dentro do `if` que verifica se a instância `p` é da classe `Funcionario` no método `calculaEmpréstimo` da classe `EmprestimoBancarioComCast` (figura 8.18)? Explique.

Exercício 8.23: ★★

O que acontecerá se o argumento para o método `calculaEmpréstimo` da classe `EmprestimoBancarioComCast` (figura 8.18) for declarado como uma instância de `Object` em vez de `Pessoa`? Explique.

Exercício 8.24: ★★

Podemos fazer um *cast* indireto de classes para transformar uma instância da classe `AutomovelDeLuxo` em uma instância da classe `Object` e usar esta para criar uma instância da classe `String`, por exemplo? Explique.

Exercício 8.25: ★★★

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 8.26: ★★★

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 8.25.

Exercício 8.27: ★★★

Usando o exercício 8.11 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 8.28: ★★★

Escreva uma classe `VeiculoAVenda` que represente informações básicas sobre um veículo genérico que esteja à venda, como `tipo` (podendo ser representado por uma string), `ano` e `preçoDeVenda`. Escreva uma classe `AutomovelAVenda` que herde de `VeiculoAVenda` e que inicialize o campo `tipo` com o valor constante "`Automovel`", e uma classe `MotocicletaAVenda` que herde de `VeiculoAVenda` e que inicialize o campo `tipo` com o valor constante "`Motocicleta`". Escreva também um programa que demonstre o uso das diversas classes.

Exercício 8.29: ★★★

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 2.22, 2.23 e 2.24.

Exercício 8.30: ★★★

Usando a classe `Ponto2D` (figura 3.4), escreva duas versões da classe `Ponto3D`: uma usando o mecanismo de delegação e outra usando o mecanismo de herança.

Exercício 8.31: ★★★

Escreva a classe `ObjetoGeometrico` que represente um objeto geométrico em duas dimensões. Essa classe deve ter métodos para inicializar o objeto, mostrar seus dados e calcular e retornar sua área e perímetro. Usando essa classe como base, escreva as classes herdeiras `Circulo` (contendo duas coordenadas para o centro e um raio), `Retangulo` (contendo dois valores para os lados) e `Triangulo` (contendo três valores para os lados), que sobrepõem os métodos descritos em `ObjetoGeometrico`. *Dicas:* A área de um círculo pode ser calculada com `Math.PI*r*r`, onde `r` é o raio do círculo. O perímetro de um círculo é dado por `2*Math.PI*r`. A área do retângulo é dada por `b*h`, onde `b` é um dos lados e `h` é o outro lado. Seu perímetro é dado por `2*b+2*h`. A área de um triângulo é dada por `Math.sqrt(s*(s-a)*(s-b)*(s-c))`, onde `Math.sqrt` é a função que calcula a raiz quadrada, `a`, `b` e `c` são os lados do triângulo, e `s` é a metade do perímetro do triângulo. O perímetro do triângulo é calculado como `(a+b+c)`.

Exercício 8.32: ★★★

Usando o exercício 8.31 como base, escreva as classes `Quadrado`, que herda da classe `Retangulo` mas somente precisa inicializar um dos lados, e as classes `TrianguloEquilatero`, `TrianguloIsosceles` e `TrianguloEscaleno`, que precisam inicializar somente um, dois ou três lados do triângulo. Para cada uma dessas classes, quais métodos devem ser sobrepostos e quais podem ser aproveitados?

8.2 Exercícios complementares do Capítulo 8

Exercício 8.33: ★

Explique com suas palavras por que podemos chamar implicitamente o método `toString` das instâncias da classe `Data` mas precisamos chamar explicitamente o mesmo método da classe ancestral dentro do método `toString` da classe `Funcionario` (figura 8.8).

Exercício 8.34: ★

Modifique o método `toString` da classe `RegistroAcademicoDeGraduacao` (figura 8.2) para que o nome do curso seja impresso em vez do seu código. Use, para isso, as constantes definidas na classe `RegistroAcademicoDeGraduacao`.

Exercício 8.35: ★

Escreva uma aplicação que use instâncias da classe `Pessoa` (figura 8.7) e tente modificar diretamente os campos dessa classe. O que acontece?

Exercício 8.36: ★

Por que, dentro do método `quantoCusta` da classe `Automovel` (figura 8.11), a variável `preço` deve ser inicializada com zero?

Exercício 8.37: ★

O valor 0 (zero) é usado para inicializar a variável `preço` no método `quantoCusta` da classe `Automovel` (figura 8.11). Explique, com suas palavras, por que essa constante não precisa ter o sufixo `f` (para explicitar o tipo `float`).

Exercício 8.38: ⋆

O método `imprime` da classe `ConcessionariaDeAutomoveis` (figura 8.15) faz chamadas repetidas a um método da classe `Automovel`. Otimize o método `imprime` para minimizar o número de chamadas.

Exercício 8.39: ⋆⋆

Escreva uma aplicação em Java que declare e use algumas instâncias das classes `Pessoa`, `Funcionario` e `ChefeDeDepartamento`. Veja as classes nas listagens nas figuras 8.7, 8.8 e 8.9 como referências.

Exercício 8.40: ⋆⋆

Os métodos `quantoCusta` e `toString` na classe `Automovel` (figura 8.11) não prevêem o caso de alguma instância da classe ser criada com um valor para o combustível diferente dos previstos. Modifique os métodos para que estes possam tratar de outros valores não-previstos, considerando que, se o valor do campo `combustível` não for um dos quatro valores válidos, o combustível será considerado como sendo gasolina.

Exercício 8.41: ⋆⋆

Modifique a classe `AutomovelBasico` (figura 8.12) para que esta tenha constantes finais booleanas `COMRETROVISOR`, `SEMRETROVISOR`, `COMLIMPADOR` etc., de forma similar à classe `Automovel` e suas constantes para representar tipos de combustível. Faça o mesmo para a classe `AutomovelDeLuxo`, criando nesta as constantes `COMDIREÇÃOHIDRÁULICA` etc.

Exercício 8.42: ⋆⋆⋆

Escreva um método `comQueIdadeEntrou` na classe `Funcionario` (figura 8.8) que retorne a idade aproximada do funcionário quando ele foi admitido. Note que para isso talvez seja necessário escrever métodos na classe `Data` que permitam o acesso ao seu campo `ano`.

Exercício 8.43: ⋆⋆⋆

Modifique a classe `ConcessionariaDeAutomoveis` (figura 8.15) para que um pequeno menu com os dados básicos (modelo e cor) de cada automóvel seja apresentado, permitindo ao usuário escolher um valor correspondente ao automóvel que o interessa. Quando o usuário entrar um valor, os dados completos do automóvel correspondente ao valor devem ser mostrados. *Dica:* Vale a pena escrever um método `dadosBásicos` na classe `Automovel` que retorne, como uma string, somente o modelo e cor do automóvel.

Capítulo 9

Classes abstratas e interfaces

9.1 Exercícios do Capítulo 9

Exercício 9.1: ⋆

Escreva um método `recarrega` para a classe `RoboABateria` (figura 9.3) que recarregue o robô, isto é, que acumule ao seu campo `energia` um valor passado como argumento para o método.

Exercício 9.2: ⋆

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

```
class Produto
  {
  private String identificação;
  private double custoDeFabricação;
  Produto(String i,double c)
    {
    identificação = i; custoDeFabricação = c;
    }
  abstract public String toString();
  abstract public void novoCusto(double nc);
  }
```

Exercício 9.3: ⋆

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

```
abstract class Dispositivo
  {
  private String nome;
  private long capacidadeEmBytes;
  Dispositivo(String n,long c)
    {
    nome = n;  capacidadeEmBytes = c;
    }
  abstract public String toString();
  abstract public double capacidadeEmMegabytes();
  }

class DiscoOtico extends Dispositivo
  {
  DiscoOtico(long c)
    {
    super("Disco Ótico",241172480L);
    }
  public String toString()
    {
    return "Dispositivo:"+nome+" Capacidade:"+c;
    }
  }
```

Exercício 9.4: ★

O que ocorreria se, na declaração da classe `RoboAbstrato` (figura 9.1), a palavra-chave `abstract` fosse omitida? Explique.

Exercício 9.5: ★

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

Exercício 9.6: ★

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

Exercício 9.7: ★

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

Exercício 9.8: ★

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

Exercício 9.9: ★

O que aconteceria se tentássemos chamar o método `imprimeTodosOsDados` da classe `DemoObjetosGeometricos` (figura 9.8) passando para ele, como argumento, uma string? Explique.

Exercício 9.10: ★

O que aconteceria se no método `main` da classe `DemoObjetosGeometricosEPolimorfismo` (figura 9.9) escrevêssemos a linha `o1 = new ObjetoGeometrico();` depois da declaração da referência `o1`? Explique.

Exercício 9.11: ★★

A classe `RoboSimples` (figura 9.2) representa um robô que pode se movimentar apenas nas quatro direções cardeais, mas não impõe restrições no valor do ângulo que pode ser passado para o método `mudaDireção` (herdado da classe `RoboAbstrato`). Escreva, para a classe `RoboSimples`, o método `mudaDireção`, sobreponto o método herdado, de forma que a direção atual seja calculada como:

- Se valores abaixo de 45 graus ou acima de 315 graus forem passados como argumento para o método, a direção do robô será 0 (zero) grau;
- Se valores entre 45 e 135 graus forem passados como argumento para o método, a direção do robô será 90 graus;
- Se valores entre 135 e 225 graus forem passados como argumento para o método, a direção do robô será 180 graus;
- Se valores entre 225 e 315 graus forem passados como argumento para o método, a direção do robô será 270 graus.

Exercício 9.12: ★★

Modifique o método `move` da classe `RoboABateria` (figura 9.3) para que a energia gasta para movimentos na diagonal (45, 135, 225 e 315 graus) seja igual a 14 vezes o número de passos, sendo que para os movimentos na horizontal ou vertical deve continuar igual a 10 vezes o número de passos.

Exercício 9.13: ★★

Podemos declarar um método em uma interface como `private`, `static` ou `final`? O que ocorrerá? Explique cada um dos casos.

Exercício 9.14: ★★

Reescreva a classe `ConstantesMatematicas` (mostrada na figura 5.5) como uma interface. Demonstre seu uso através de uma classe que implementa essa interface e tem acesso aos seus campos.

Exercício 9.15: ★★

Escreva a classe `Quadrado`, que implementa a interface `ObjetoGeometrico`. Use como referência as classes `Circulo` e `Retangulo` (listagens nas figuras 9.6 e 9.7).

Exercício 9.16: ★★

Escreva, para a classe `Retangulo` (figura 9.7), o método `calculaAchatamento`, que calcula o achatamento do retângulo (o achatamento é dado dividindo-se o comprimento do lado menor pelo comprimento do lado maior).

Exercício 9.17: ★★

Usando a interface `ObjetoGeometrico` (figura 9.5) como base, crie a interface `ObjetoTridimensional` que deverá ser implementada por todas as classes que representem objetos tridimensionais. A interface deve declarar os métodos `centro` (que deve retornar uma instância da classe `Ponto3D`), `calculaSuperfície`, que calcule e retorne a área de superfície do objeto tridimensional, e `calculaVolume`, que calcule e retorne o volume do objeto tridimensional.

Exercício 9.18: ★★

Inclua, na interface `ObjetoGeometrico` (figura 9.5), 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 9.19: ★★

Podemos ter na interface `Escalavel` um método `reduz` que chame o método `amplia` com valores entre zero e um? Explique.

Exercício 9.20: ★★

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

```
1  interface Resetavel
2    {
3    void reseta();
4    }
5
6  interface Modificavel
7    {
8    void reseta(int origem);
9    void modifica(int tamanho);
10   }
11
12 class Contador implements Resetavel,Modificavel
13   {
14   int valor;
15   void reseta()
16     {
17     tamanho = 1;
18     }
19   void modifica(int tam)
20     {
21     tamanho = tam;
22     }
23   }
```

Exercício 9.21: ★★★

Escreva uma classe `RoboComMemoria` que herde da classe `RoboAbstrato` e que seja capaz de armazenar o número de passos dados em cada direção cardeal. Escreva, para esse robô, um método `retornaÀOrigem` que, usando os passos dados em cada direção e a posição atual, calcule a sua origem. *Dica:* Se o robô somente se move nas direções cardeais, não é necessário armazenar quatro valores para a memória do movimento, pois cada passo dado em uma direção corresponde ao valor negativo desses passos dados na direção oposta.

Exercício 9.22: ★★★

Escreva uma classe `RoboPesadoABateria` que herde da classe `RoboABateria` e que represente também o peso do robô em quilos. Esse peso determinará a constante a ser usada para o gasto de energia com o movimento do robô: para que o robô se movimente na direção horizontal ou vertical, o gasto de energia será o peso do robô vezes o número de passos a ser percorrido, e para que o robô se movimente na diagonal, o gasto de energia será 1.4 multiplicado pelo peso do robô multiplicado pelo número de passos a ser percorrido. Veja também o exercício 9.12.

Exercício 9.23: ★★★

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` implemente também a classe `Cor`, que contém constantes representando diversas cores padrão.

Exercício 9.24: ★★★

Escreva para a interface `ObjetoGeometrico` (figura 9.5) 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`.

Exercício 9.25: ★★★

Escreva a classe `Triangulo`, que implementa a interface `ObjetoGeometrico`. Use como referência as classes `Circulo` e `Retangulo` (listagens nas figuras 9.6 e 9.7). *Dica:* Para calcular o perímetro e a área de um triângulo qualquer, devemos obter os lados a, b e c do triângulo. Seu perímetro será calculado como $a+b+c$ e sua área como $\sqrt{s(s-a)(s-b)(s-c)}$, onde s é a metade do perímetro. A raiz quadrada pode ser calculada com o método `Math.sqrt`. Para obter os lados, deve-se tomar os pontos do triângulo dois a dois e calcular o lado com a fórmula $l_{AB} = \sqrt{(x_A - x_B)^2 + (y_A - y_B)^2}$, onde l_{AB} é o lado que une os pontos A e B.

Exercício 9.26: ★★★

Considere a classe `DemoObjetosGeometricos` (figura 9.8). Modifique o seu método `imprimeTodosOsDados` para que este imprima também o achatamento dos retângulos (veja exercício 9.16).

Exercício 9.27: ★★★

Escreva a classe `Esfera`, que implemente a interface `ObjetoTridimensional` (exercício 9.17) e represente uma esfera no espaço tridimensional. O volume de uma esfera é igual a $\frac{4}{3}\pi r^3$, onde r é o raio da esfera, e a área de sua superfície é igual a $4\pi r^2$.

Exercício 9.28: ★★★

Escreva a classe `Paralelepipedo`, que implemente a interface `ObjetoTridimensional` (exercício 9.17) e represente um paralelepípedo no espaço tridimensional. O volume de um paralelepípedo é igual a $a \times b \times c$, onde a, b e c são o comprimento de seus lados. A área da superfície é dada por $2(ab+bc+ac)$. A classe deve usar duas instâncias da classe `Ponto3D` para representar os pontos opostos do paralelepípedo. Os comprimentos dos lados podem ser obtidos a partir das diferenças das coordenadas x, y e z dos cantos opostos. Veja também a classe `Retangulo` (figura 9.7) como exemplo.

Exercício 9.29: ★★★

Crie a classe `RetanguloEscalavel` que implementa simultaneamente as interfaces `ObjetoGeometrico` e `Escalavel`. Para simplificar o cálculo do novo tamanho do retângulo, considere que um dos pontos que o representa fica fixo e somente o outro tem seu tamanho modificado.

Exercício 9.30: ★★★★

Usando o resultado do exercício 9.21 e a classe `RoboABateria` (figura 9.3) como base, escreva uma classe `RoboQueDeveVoltar` que implemente o mecanismo de retorno à posição de origem do robô e o mecanismo de consumo de energia dado. Escreva o método `move` de forma que o robô só se moverá se ainda houver energia suficiente para ele voltar ao ponto de origem. *Dica:* Existem duas soluções para implementar o mecanismo de volta, sendo uma delas consideravelmente mais econômica (em termos de uso de energia do robô) do que a outra – tente implementar as duas soluções.

9.2 Exercícios complementares do Capítulo 9

Exercício 9.31: ★

Modifique o construtor da classe `Circulo` (figura 9.6) para que nenhum círculo com raio negativo possa ser criado. Caso um valor negativo seja passado como raio, o construtor deve considerar o raio como sendo zero.

Exercício 9.32: ★

Escreva um método `quantaEnergia` para a classe `RoboABateria` (figura 9.3) que retorne a quantidade de energia restante para o robô.

Exercício 9.33: ★

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

Exercício 9.34: ★

No método `toString` da classe `LivroDeBiblioteca` (figura 9.15), usamos os campos `localização` e `estáEmprestado` em vez dos métodos que permitem o acesso a esses campos. Existe alguma vantagem ou desvantagem em fazer isso? Explique.

Exercício 9.35: ★

Explique, com suas palavras, por que não faz sentido declararmos a classe `Livro` (figura 9.14) como implementando a interface `ItemDeBiblioteca` (figura 9.13).

Exercício 9.36: ★

O campo `empréstimoPodeSerRenovado`, na interface `ItemRaroDeBiblioteca` (figura 9.17) não foi utilizado na classe `MapaDeBiblioteca` (figura 9.19), que implementa a interface `ItemRaroDeBiblioteca`. Em que métodos da classe `MapaDeBiblioteca` esse campo pode ser usado adequadamente?

Exercício 9.37: ★★

Escreva um método `podeSeMover` para a classe `RoboABateria` (figura 9.3) que retorne `true` caso o robô possa se mover na distância passada como argumento para o método, e `false` caso contrário. Escreva outro método sobrecarregado que não receba argumentos e considere que a distância a ser percorrida é um.

Exercício 9.38: ★★

Escreva a classe `FitaDeVideo`, que encapsula os dados de uma fita de vídeo educativo. Escreva também a classe `FitaDeVideoDeBiblioteca` que herda da classe `FitaDeVideo` e implementa a interface `ItemDeBiblioteca`. Use como referência a classe `LivroDeBiblioteca`, na figura 9.15.

Exercício 9.39: ★★★

Modifique a classe abstrata `RoboAbstrato` (figura 9.1) de forma que em vez de usar dois valores separados para a posição do robô, a classe use uma instância da classe `Ponto2D`. Que modificações deverão ser feitas nas classes herdeiras?

Exercício 9.40: ★★★

Modifique o método `amplia` da classe `RetanguloEscalavel` para que os dois pontos que definem o retângulo tenham suas posições modificadas. A figura 9.1 mostra duas maneiras de se modificar a escala de um retângulo, sendo que a primeira (ao centro) modifica somente um ponto, mantendo o outro fixo, enquanto a segunda (à esquerda) modifica os dois pontos.

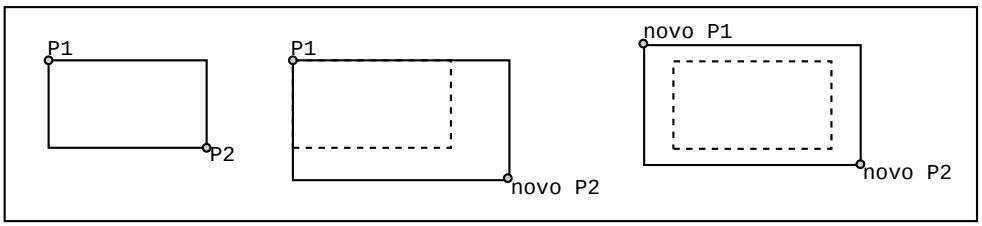


Figura 9.1: Duas maneiras de se modificar a escala de um retângulo

Capítulo 10

Pacotes de classes em Java

10.1 Exercícios do Capítulo 10

Exercício 10.1: ⋆

Explique o que aconterceria se os campos `hora`, `minuto` e `segundo` da classe `Hora` (figura 10.2) fossem declarados como `protected`. Considere o que aconteceria com as classes `DataHora` e `DemoDataHora` (listagens nas figuras 10.3 e 10.4).

Exercício 10.2: ⋆

O que aconteceria se a linha `import DataHora.*;` da classe `DemoDataHora` (figura 10.4) fosse escrita como `import DataHora.DataHora;`?

Exercício 10.3: ⋆

Por que a classe `DemoDataHora` (figura 10.4) não precisou ser declarada como `public`?

Exercício 10.4: ⋆

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

```
1  /* Esta classe faz parte de todos os pacotes em br.universidade.bioinfo */
2  package br.universidade.bioinfo.*;
3
4  class ExtremamenteGenerica
5    {
6    private int valor;
7    }
```

Exercício 10.5: ⋆

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

```
1  import *;
2
3  class Teste
4    {
5    public static void main(String[] args)
6      {
7      Ponto2D p = new Ponto2D(0,0);
8      Pessoa m = new Pessoa("Cay Horstmann",124123,
9                            new Data((byte)12,(byte)7,(short)1953));
10     }
11   }
```

Exercício 10.6: ⋆⋆

Identifique e explique o(s) erro(s) nas classes abaixo. Considere que as duas classes devam coexistir no mesmo ambiente de desenvolvimento ou execução.

```
1  package br.universidade.bioinfo.Geometria;
2
3  public class Ponto2D
4    {
5    protected double x,y;
6    public Ponto2D(double _x,double _y)
7      {
8      x = _x; y = _y;
9      }
10   }
```

```
1  package br.universidade.bioinfo.Geometria;
2
3  public class Ponto3D extends Ponto2D
4    {
5    protected double z;
6    public Ponto3D(double _x,double _y,double _z)
7      {
8      x = _x; y = _y; z = _z;
9      }
10   }
```

Capítulo 11

Arrays em Java

11.1 Exercícios do Capítulo 11

Exercício 11.1: ⋆

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 11.2: ⋆

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 11.3: ⋆

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

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

Exercício 11.4: ★

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

```
class ClasseMisteriosa
  {
  public static void main(String[] argumentos)
    {
    double[] valores = {1,2,3,4,5,6};
    double[] primeiraCópia = valores;
    double[] segundaCópia = valores;
    primeiraCópia[1] = 1;
    segundaCópia[2] = valores[0]+primeiraCópia[1];
    primeiraCópia[3] = valores[1]+segundaCópia[2];
    valores[4] = primeiraCópia[2]+segundaCópia[3];
    valores[5] = segundaCópia[3]+primeiraCópia[4];
    }
  }
```

Exercício 11.5: ★

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

```
class ClasseMisteriosa
  {
  public static void main(String[] argumentos)
    {
    float[] constantes = {100f,10f,1f,0.1f,0.01f,0.001f};
    float[] duplicata = constantes;
    resetaArray(duplicata);
    }
  private static void resetaArray(float[] array)
    {
    for(int índice=0;índice<array.length;índice++)
      array[índice] = 0f;
    }
  }
```

Exercício 11.6: ★

O método abaixo pode ser um método da classe `ArrayDeFloats` (figura 11.4)? Explique.

```
public void mudaTamanho(int novoTamanho)
  {
  array.length = novoTamanho;
  }
```

Exercício 11.7: ★

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 11.8: ★

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 11.9: ★

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.

Exercício 11.10: ★

Usando a classe `CalculoPiQuadradoSobre6` (figura 11.3) como base, escreva uma aplicação em Java que calcule os termos da série que converge para $3/4$ usando um array. Veja também o exercício 7.29.

Exercício 11.11: ★

Modifique a classe `ArrayDeFloats` (figura 11.4), 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 11.12: ★

Modifique o método `toString` da classe `ArrayDeObjetosGeometricos` (figura 11.7) para que somente os elementos do array encapsulado que forem diferentes de `null` sejam impressos.

Exercício 11.13: ★

Escreva para a classe `MatrizDeDoubles` métodos que permitam o acesso e a modificação dos valores individuais da matriz encapsulada. *Dica:* Veja a classe `ArrayDeFloats` (figura 11.4).

Exercício 11.14: ★

Escreva para a classe `MatrizDeDoubles` o método `éQuadrada`, que retorna `true` se a matriz for quadrada (isto é, tem o mesmo número de linhas e colunas).

Exercício 11.15: ★★

Usando a classe `CalculoPiQuadradoSobre6` (figura 11.3) como base, escreva uma aplicação em Java que calcule os termos da série que converge para o logaritmo de 2 na base natural usando um array. Veja também o exercício 7.32.

Exercício 11.16: ★★

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 11.9 como base.

Exercício 11.17: ★★

Crie na classe `ArrayDeFloats` (figura 11.4) 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 11.18: ★★

Crie na classe `ArrayDeFloats` (figura 11.4) 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 11.19: ★★

Crie na classe `ArrayDeFloats` (figura 11.4) um método `total` que some todos os valores do array, retornando o resultado dessa somatória.

Exercício 11.20: ★★

Escreva um método `imprimeChefes` para a classe `Equipe` (figura 11.6) que receba o array de instâncias da classe `Funcionario` como argumento e imprima somente os elementos que forem instâncias da classe `ChefeDeDepartamento`. O método não deverá imprimir instâncias que sejam iguais a `null`.

Exercício 11.21: ★★

Crie na classe `MatrizDeDoubles` (figura 11.12) o método `toString` que retorne uma string contendo os elementos da matriz. Faça com que esta string tenha várias linhas, correspondente às linhas da matriz. *Dica:* O caracter `'\n'` significa quebra de linha, e pode estar presente no meio de strings.

Exercício 11.22: ★★

Crie na classe `MatrizDeDoubles` (figura 11.12) 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 11.23: ★★

Crie na classe `MatrizDeDoubles` (figura 11.12) um método `soma` que some (acumule) aos elementos da matriz encapsulada uma constante do tipo `double` que será passada como argumento.

Exercício 11.24: ★★

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 11.25: ★★

Usando o exercício 11.24 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 11.26: ★★

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 11.27: ★★

Crie na classe `ArrayDeFloats` o método `média`, que retorna a média dos valores encapsulados no array. *Dica:* Valores infinitos, se houver, não devem ser considerados.

Exercício 11.28: ★★

Crie na classe `ArrayDeFloats` (figura 11.4) um método `soma` que receba uma outra instância da classe `ArrayDeFloats` como argumento e acumule os valores do array passado como argumento ao array encapsulado. Essa operação somente poderá ser feita se os arrays tiverem o mesmo tamanho; se o tamanho for diferente, o array encapsulado não deverá ser modificado. Veja também o exercício 11.18.

Exercício 11.29: ★★

Considere a classe `ArrayDeFloats` (figura 11.4). Crie nessa classe o método `troca` que recebe dois valores inteiros como argumentos e troca os valores nas posições especificadas pelos argumentos. Por exemplo, se os valores do array encapsulado forem $\{3.6, 2.7, 8.0, 9.2, 1.5, 4.1\}$ e o método `troca` for chamado com os argumentos 1 e 4, os elementos nesses índices serão trocados de forma que os valores do array encapsulado serão $\{3.6, 1.5, 8.0, 9.2, 2.7, 4.1\}$ (note os valores que foram trocados). *Dica:* É possível fazer essa troca usando um valor temporário do tipo `float`.

Exercício 11.30: ★★

Usando o exercício 11.29 como base, reescreva o método `troca` para que este somente troque os valores se o valor na posição do primeiro índice for **menor** do que o valor na posição do segundo índice. Por exemplo, se os valores no array encapsulado forem $\{3.6, 2.7, 8.0, 9.2, 1.5, 4.1\}$, uma chamada ao método `troca(0,1)` trocaria os valores nas posições 0 e 1, mas uma chamada ao método `troca(1,2)` não trocaria os valores de índices 1 e 2 de posição.

Exercício 11.31: ★★

Escreva para a classe `MatrizDeDoubles` (figura 11.12) o método `éIdentidade` que retorne `true` se a matriz for uma matriz identidade (isto é, se a matriz for quadrada, com todos os elementos da diagonal iguais a um, e com todos os outros elementos iguais a zero). *Dica:* Use o método `éQuadrada`, criado como resposta ao exercício 11.14.

Exercício 11.32: ★★

Escreva para a classe `MatrizDeDoubles` o método `todosElementosSãoIguais` que retorne `true` se todos os elementos da matriz tiverem o mesmo valor. *Dica:* É possível escrever este método de maneira simplificada se antes criarmos um método `menorValor` que retorne o menor valor da matriz, nos mesmos moldes do método `menorValor` que já existe na classe.

Exercício 11.33: ★★

Escreva uma classe `MatrizDeCaracteres` que encapsule uma matriz de caracteres (valores do tipo `char`). As dimensões dessa matriz devem ser especificadas através do construtor da classe. O construtor da classe deve também inicializar a matriz com caracteres aleatórios (que podem ser obtidos usando-se a expressão `(char)('A'+Math.random()*26)`). Escreva também um método `toString` para essa classe.

Exercício 11.34: ★★

Escreva uma classe `ArrayDeDatas`, com a mesma funcionalidade da classe `ArrayDeObjetosGeometricos` (figura 11.7), que encapsule um array de instâncias da classe `Data`.

Exercício 11.35: ★★★

Escreva na classe `ArrayDeFloats` o método `maisPróximo`, que recebe um valor do tipo `float` como argumento e retorna o valor do array encapsulado que seja mais próximo (ou seja, cuja diferença seja a menor) do valor passado. Por exemplo, se o array encapsulado for $\{0,-2,-4,10\}$ e o argumento para o método for 6, o método deverá retornar 10. *Dica:* Valores infinitos (positivos ou negativos) não devem ser considerados.

Exercício 11.36: ★★★

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 11.37: ★★★

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 11.36.

Exercício 11.38: ★★★

Escreva na classe `ArrayDeFloats` o método `éCrescente`, que verifica se os elementos de um array estão ordenados crescentemente, comparando cada elemento do array com seu próximo, retornando `true` se todos os elementos forem menores que os seus respectivos próximos ou `false` se qualquer um for maior do que o próximo.

Exercício 11.39: ★★★

Crie na classe `ArrayDeFloats` (figura 11.4) um método `produtoEscalar` que retorne um valor do tipo `float` que seja o produto escalar do array encapsulado e de uma outra instância de `ArrayDeFloats` passada como argumento. Por exemplo, se o array encapsulado for $\{9,2,-6,7,0\}$ e o passado como argumento for $\{1,-4,5,9,2\}$, o produto escalar será $9\times1-2\times4-6\times5+7\times9+0\times2=34$. O método deverá retornar a constante `Float.NaN` caso os arrays sejam de tamanhos diferentes.

Exercício 11.40: ★★★

Crie na classe `ArrayDeFloats` o método `existeQualquer`, que recebe uma outra instância da classe `ArrayDeFloats` como argumento e retorna o booleano `true` se qualquer um dos elementos do array passado como argumento existir no array encapsulado.

Exercício 11.41: ★★★

Crie na classe `ArrayDeFloats` o método `existemTodos`, que recebe uma outra instância da classe `ArrayDeFloats` como argumento e retorna o booleano `true` se todos os elementos do array passado como argumento existirem no array encapsulado, em qualquer ordem e independentemente de repetições.

Exercício 11.42: ★★★

Crie na classe `ArrayDeFloats` (figura 11.4) um método `distânciaEuclideana` que retorne um valor do tipo `float` que seja igual à distância euclideana acumulada entre o array encapsulado e de uma outra instância de `ArrayDeFloats` passada como argumento. A distância euclideana entre dois arrays numéricos é definida como sendo a raiz quadrada das somas dos quadrados das diferenças de seus elementos. Por exemplo, se o array encapsulado for $\{1,3,0\}$ e o passado como argumento for $\{1,9,-4\}$, a distância euclideana será igual a $\sqrt{(1-1)^2+(3-9)^2+(0-(-4))^2} = \sqrt{0+36+16} \approx 7.2111$. O método deverá retornar a constante `Float.NaN` caso os arrays sejam de tamanhos diferentes. *Dica:* A raiz quadrada de uma expressão qualquer pode ser calculada com o método `sqrt` da classe `Math`, que recebe como argumento um valor ou expressão e retorna a raiz quadrada daquele argumento.

Exercício 11.43: ★★★

Crie na classe `MatrizDeDoubles` (figura 11.12) um método `soma` que receba uma outra instância da classe `MatrizDeDoubles` como argumento e acumule os valores da matriz passada como argumento à matriz encapsulada. Essa operação somente poderá ser feita se as matrizes tiverem o mesmo tamanho; se o tamanho for diferente, a matriz encapsulada não deverá ser modificada. Veja também os exercícios 11.28 e 11.23.

Exercício 11.44: ★★★

Escreva para a classe `MatrizDeDoubles` (figura 11.12) o método `éTriangularSuperior` que retorne `true` se a matriz for triangular superior, isto é, se os elementos nas posições (l,c) forem iguais a zero para $l > c$, onde l é o índice da linha da matriz e c o índice da coluna da matriz.

Exercício 11.45: ★★★

Escreva para a classe `MatrizDeDoubles` (figura 11.12) o método `todosElementosSãoDiferentes` que retorne `true` se todos os elementos da matriz forem diferentes, isto é, se não existirem elementos duplicados na matriz. Note que esse método não é o inverso lógico do método `todosElementosSãoIguais`, pedido no exercício 11.32.

Exercício 11.46: ★★★

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 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.

Exercício 11.47: ★★★

Escreva a classe que implementa um *array de escrita única* (exercício 11.46) como sendo uma classe herdeira da classe `ArrayDeFloats`. Quais métodos devem ser sobrepostos?

Exercício 11.48: ★★★

Usando o exercício 11.46 como base, crie uma classe que implemente um array cujos elementos só podem ser modificados um certo número 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 deverá poder ser modificado o número de vezes que for especificado.

Exercício 11.49: ★★★

Escreva para a classe `ArrayDeObjetosGeometricos` (figura 11.7) os métodos `calculaMaiorÁrea` e `calculaMenorÁrea` que calculem e retornem, respectivamente, a maior e menor área dos objetos contidos no array encapsulado. *Dica:* Verifique se o elemento é `null` antes de tentar calcular a sua área. Veja também a classe `ArrayDeFloats` (figura 11.4) como referência.

Exercício 11.50: ★★★

Crie uma classe `EntradaEmAgenda` que contenha os dados necessários para armazenar uma entrada de agenda (hora, dia, mês, ano, assunto). Crie nessa classe, além do construtor e do método `toString`, um método `éNoDia` que recebe valores de dia, mês e ano e retorna `true` se o dia, mês e ano daquela instância da classe forem iguais aos argumentos passados. Crie um método similar chamado `éNoMês` que recebe valores de mês e ano somente e retorna `true` se o mês e ano daquela instância da classe forem iguais aos argumentos passados.
Crie também uma classe `Agenda` que encapsule uma agenda de compromissos, que será representado por um array de instâncias da classe `EntradaEmAgenda`. Crie um método `listaDia` que recebe valores de dia, mês e ano e lista **todas** as instâncias de `EntradaEmAgenda` que caem naquele dia, mês e ano, e um método `listaMês` que recebe valores de mês e ano e lista **todas** as instâncias de `EntradaEmAgenda` que caem naquele mês e ano.

Exercício 11.51: ★★★

Escreva uma classe `Banco` que encapsule um array de instâncias da classe `ContaBancariaSimplificada` (exercício 2.39). Escreva, para essa classe, um método `total` que calcule e retorne o total dos saldos de todas as contas bancárias encapsuladas como elementos do array.

Exercício 11.52: ★★★

Usando a classe `CalculadoraDeLinhaDeComando` (figura 11.17) 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 11.53: ★★★

Escreva uma classe `ArrayDeFloatsUnicos` que herde da classe `ArrayDeFloats` e implemente um array onde valores repetidos não podem existir. *Dica:* Cada vez que tentarmos inserir um valor nesse array, o método `modifica` deve verificar se o valor existe, só modificando ou inserindo se ainda não existir no array encapsulado.

Exercício 11.54: ★★★

Escreva uma classe `ArrayDePontos2D`, no molde da classe `ArrayDeFloats` (figura 11.4) que encapsule um array de instâncias da classe `Ponto2D` (figura 3.4). Essa classe deve ter os seguintes métodos:

- Construtor, que recebe como argumento um número máximo de instâncias da classe `Ponto2D` que serão encapsuladas pela classe,
- `tamanho`, que retorna o tamanho do array encapsulado,
- `modifica`, que recebe como argumentos um valor inteiro (posição) e uma instância da classe `Ponto2D`, e faz com que a instância naquela posição do array passe a ser a passada como argumento,
- `valor`, que recebe como argumento um valor inteiro (posição) e retorna a instância armazenada naquela posição do array,
- `toString`, que retorna uma única string contendo todas os valores das instâncias no array encapsulado na classe.

Para facilitar a criação da classe, considere que a classe `Ponto2D` tem ao menos um construtor que recebe argumentos (veja o exercício 4.19) e métodos para recuperar as coordenadas x e y do ponto.

Exercício 11.55: ★★★

Escreva para a classe `ArrayDePontos2D` (exercício 11.54) um método `pontoSuperior` que retorne o ponto cuja coordenada `y` é a maior dentre as encapsuladas no array; um método `pontoInferior` que retorne o ponto cuja coordenada `y` é a menor dentre as encapsuladas no array; um método `pontoMaisÀEsquerda` que retorne o ponto cuja coordenada `y` é a menor dentre as encapsuladas no array; e um método `pontoMaisÀDireita` que retorne o ponto cuja coordenada `y` é a maior dentre as encapsuladas no array.

Exercício 11.56: ★★★

Escreva para a classe `ArrayDePontos2D` um método `retânguloEnvolvente` que retorne uma instância da classe `Retangulo` (exercícios 2.36 e 4.23) contendo o retângulo envolvente de todos os pontos do array encapsulado, de forma que todos os pontos do array estejam localizados dentro do retângulo (veja a figura 11.1, onde o retângulo é definido pelas coordenadas $(x1, y1)$ e $(x2, y2)$). *Dica:* Use os métodos criados no exercício 11.55.

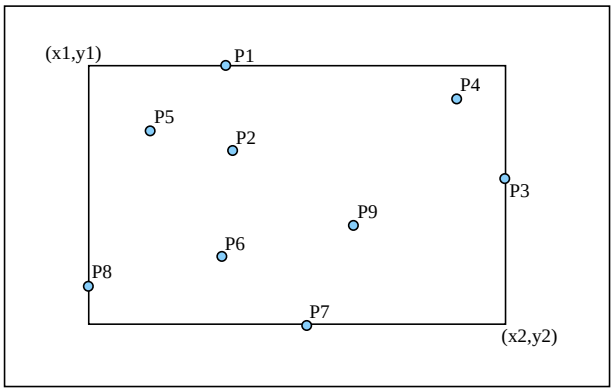


Figura 11.1: Retângulo envolvente de uma série de pontos

Exercício 11.57: ★★★

Escreva para a classe `ArrayDePontos2D` (figura 11.54) um método `centro` que calcule e retorne o ponto central de todos os pontos diferentes que estejam encapsulados no array, isto é, o ponto localizado nas médias das coordenadas x e y.

Exercício 11.58: ★★★

Escreva para a classe `ArrayDePontos2D` um método `dispersão` que calcule e retorne a dispersão de todos os pontos do array. A dispersão é calculada como a soma de todas as distâncias entre o ponto central e cada ponto do array. Use o resultado do exercício 11.57.

Exercício 11.59: ★★★

Generalize a solução do problema das oito rainhas (implementada pelas classes `Rainha` e `OitoRainhas`, nas listagens nas figuras 11.10 e 11.11, respectivamente) para resolver o problema para um número qualquer de rainhas. O tabuleiro deve ser também ajustado ao número de rainhas, isto é, se o problema for posicionar onze rainhas sem que uma ataque as outras, o tabuleiro deverá ter onze linhas e onze colunas. *Dicas:* Não existem soluções para duas e três rainhas, mas esses casos podem ser usados para testar o algoritmo. Curiosamente, o algoritmo resolve o problema das onze rainhas em menos passos (e conseqüentemente mais rapidamente) do que o das oito rainhas.

Exercício 11.60: ★★★

Modifique o algoritmo das oito rainhas para que todas as soluções sejam achadas, e não somente a primeira. As modificações no algoritmo devem incluir:

- Um laço principal que execute enquanto for possível, só parando quando a primeira rainha não puder ser mais posicionada no tabuleiro;
- Quando uma solução for encontrada, o algoritmo deve tentar continuar movendo a última rainha, e reiniciar o laço principal.

Exercício 11.61: ★★★

Escreva para a classe `ArrayDeDatas` (exercício 11.34) um método `antesDe` que receba como argumento uma instância da própria classe `Data` e retorne uma nova instância da classe `ArrayDeDatas` que contenha somente as datas que estavam no array encapsulado que sejam anteriores à data passada como argumento. *Dica:* Use o resultado do exercício 2.48.

Exercício 11.62: ★★★

Escreva uma classe `MatrizDeNumerosComplexos`, com a mesma funcionalidade da classe `MatrizDeDoubles` (figura 11.12), que encapsule uma matriz bidimensional de instâncias da classe `NumeroComplexo` (veja os exercícios 4.20 e 11.13).

Exercício 11.63: ★★★

Crie na classe `MatrizDeNumerosComplexos` (exercício 11.62) 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. Veja também o exercício 11.22.

Exercício 11.64: ★★★

Crie na classe `MatrizDeNumerosComplexos` (exercício 11.62) um método `soma` que some (acumule) aos elementos da matriz encapsulada um outro número complexo que será passado como argumento para o método.

Exercício 11.65: ★★★

Considere um bilhete de loteria instantânea (raspadinha), que contenha seis valores. 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.

Exercício 11.66: ★★★★

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. Por exemplo, a matriz

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

representa um quadrado mágico (onde não existe repetição de valores e todos estão em seqüência. Para efeitos deste exercício poderíamos considerar uma matriz onde todos os elementos fossem iguais como sendo também 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 11.67: ★★★★

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'`. 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 11.68: ★★★★

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 11.69: ★★★★

O jogo japonês *Go* 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 *Go*, 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. 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 11.67), 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 11.70: ★★★★

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` (figura 11.4) 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 $P < Q$.

Exercício 11.71: ★★★★

Escreva para a classe `ArrayDePontos2D` (figura 11.54) um método `menorDistância` que calcule e retorne a menor distância entre dois pontos diferentes que estejam encapsulados no array. Para isso use uma versão do método `distânciaEuclideana` (exercício 11.42) que considere cada ponto como um array onde os elementos são as coordenadas.

Exercício 11.72: ★★★★

Escreva para a classe `ArrayDePontos2D` (figura 11.54) um método `maisPróximos` que retorne uma nova instância da classe `ArrayDePontos2D`. Essa instância deverá conter somente duas instâncias da classe `Ponto2D`, que são os pontos mais próximos entre si. *Dica:* Veja também o exercício 11.71.

Exercício 11.73: ★★★★

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 varre 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 para indicar que este foi removido.
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
início	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 (bizarramente, a expressão `(Float.NaN == Float.NaN)` retorna `false` e a expressão `(Float.NaN != Float.NaN)` retorna `true` !).
Crie, para a classe `ArrayDeFloats` (figura 11.4), 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 array.

Exercício 11.74: ★★★★

Escreva para a classe `MatrizDeDoubles` (figura 11.12) o método `parteTriangularSuperior` que retorne uma matriz irregular contendo os valores da matriz (se esta for triangular superior) ou `null` (se a matriz não for triangular superior).

Exercício 11.75: ★★★★

Considere o método `existe`, pedido como solução do exercício 11.26. 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 11.70) 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 11.76: ★★★★

Escreva para a classe `MatrizDeDoubles` (figura 11.12) 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:

$$\begin{bmatrix} \mathtt{A}_{(0,0)} & \mathtt{A}_{(0,1)} & \mathtt{A}_{(0,2)} & \mathtt{A}_{(0,3)} \\ \mathtt{A}_{(1,0)} & \mathtt{A}_{(1,1)} & \mathtt{A}_{(1,2)} & \mathtt{A}_{(1,3)} \end{bmatrix} \begin{bmatrix} \mathtt{B}_{(0,0)} & \mathtt{B}_{(0,1)} & \mathtt{B}_{(0,2)} \\ \mathtt{B}_{(1,0)} & \mathtt{B}_{(1,1)} & \mathtt{B}_{(1,2)} \\ \mathtt{B}_{(2,0)} & \mathtt{B}_{(2,1)} & \mathtt{B}_{(2,2)} \\ \mathtt{B}_{(3,0)} & \mathtt{B}_{(3,1)} & \mathtt{B}_{(3,2)} \end{bmatrix} = \begin{bmatrix} \mathtt{C}_{(0,0)} & \mathtt{C}_{(0,1)} & \mathtt{C}_{(0,2)} \\ \mathtt{C}_{(1,0)} & \mathtt{C}_{(1,1)} & \mathtt{C}_{(1,2)} \end{bmatrix}$$

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 11.77: ★★★★

Uma das características do algoritmo de ordenação por seleção é a necessidade de marcar valores já selecionados como tendo sido removidos. Na descrição do algoritmo mostrada no exercício 11.73 usamos o valor especial `Float.NaN` para marcar os valores removidos, mas isso impossibilita a ordenação de um array que já contenha um ou mais valores `Float.NaN`.
Uma solução para esse problema seria criar um array de valores booleanos associado ao array de valores de ponto flutuante, que representaria, para cada valor, se este já foi removido (`true`) ou não (`false`). Dessa forma, qualquer valor presente no array de valores de ponto flutuante poderia ser considerado para ordenação. Escreva versões dos métodos `ordenaCrescentePorSeleção` e `ordenaDecrescentePorSeleção` que usem essa abordagem. *Dica:* Como valores `Float.NaN` não podem ser comparados com outros valores, escreva os métodos de forma que, se existirem valores `Float.NaN` no array, estes sejam colocados no final do array, mesmo que fiquem após valores infinitos positivos (`Float.POSITIVE_INFINITY`).

Exercício 11.78: ★★★★★

É possível embutir um pouco de *Inteligência Artificial* no jogo-da-velha do exercício 11.67, 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 11.2, 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 11.67 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.

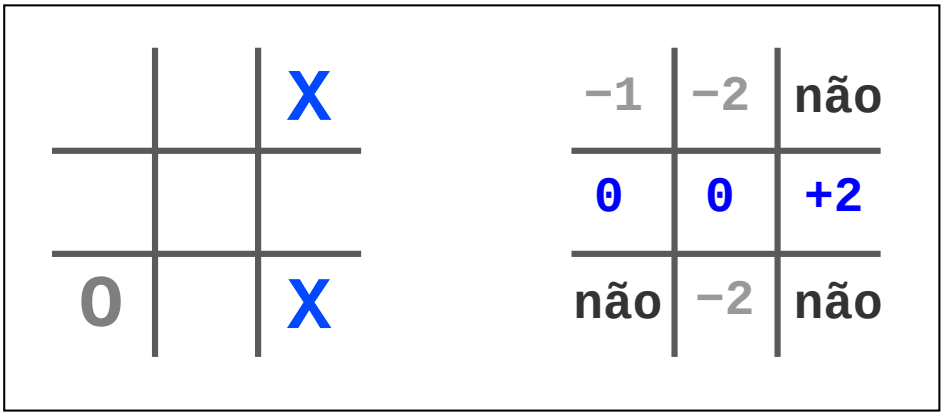


Figura 11.2: Jogo-da-velha e respectiva matriz de decisão da próxima jogada

Exercício 11.79: ★★★★★

Também é possível adaptar o jogo de *go* (exercício 11.69) 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 11.3 mostra, para uma determinada posição, quais são as posições que correspondem às vizinhanças.
Usando o exercício 11.69 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 11.78.

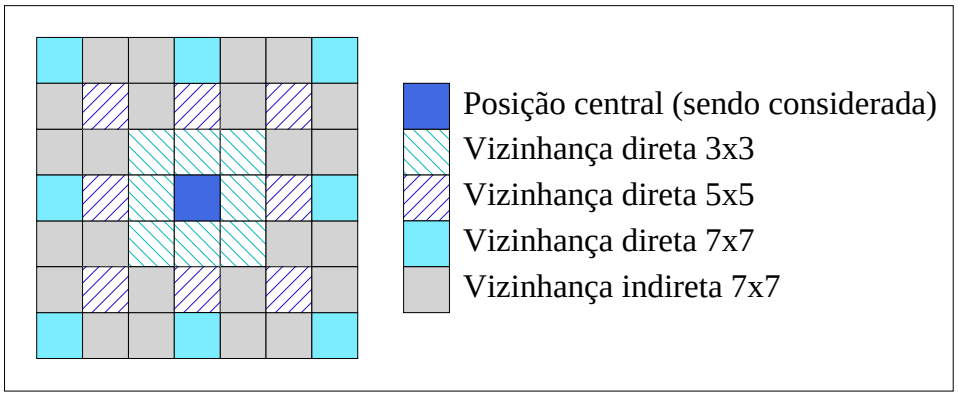


Figura 11.3: Vizinhanças para cálculo da heurística no jogo *go*

Exercício 11.80: ★★★★★

Considere os quadrados mágicos descritos no exercício 11.66. Uma aplicação poderia tentar criar quadrados mágicos de 3×3 posições com valores inteiros, não-repetidos e seqüenciais 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 cem combinações fossem testadas por segundo, este algoritmo demoraria mais de um mês 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).

11.2 Exercícios complementares do Capítulo 11

Exercício 11.81: ★

O construtor da classe `JogoDeDamas` (figura 11.13) inicializa todo o array de duas dimensões com o caracter ponto (`'.'`) para depois reinicializar algumas posições com os caracteres que representam as peças. Modifique o construtor para que este seja mais eficiente, inicializando com pontos somente as posições que não serão inicializadas posteriormente.

Exercício 11.82: ★

O que aconteceria se o laço no método `menorValor` e `maiorValor` na classe `ArrayDeFloats` (figura 11.4) iniciasse em zero em vez de um? Explique.

Exercício 11.83: ★

Quantos elementos estão contidos no array irregular mostrado na classe `TrianguloDePascal` (figura 11.16)? Quantos seriam usados se utilizássemos um array regular em vez de irregular? E se o tamanho do array fosse 1000000 linhas em vez de 10?

Exercício 11.84: ★★

Modifique o método `toString` da classe `JogoDeDamas` (figura 11.13) para que este imprima uma borda ao redor do tabuleiro. Essa borda pode ser feita com os caracteres `'-'` (para linhas horizontais), `'|'` (para linhas verticais) e `'+'` para intersecções.

Exercício 11.85: ★★

Explique, com suas palavras, por que o algoritmo de busca binária somente funciona em arrays ordenados.

Exercício 11.86: ★★★

Modifique o método `toString` da classe `JogoDeDamas` (figura 11.13) para que este imprima um tabuleiro ampliado, de forma que em vez de um caracter para representar cada peça, quatro caracteres arranjados dois a dois sejam usados. Dessa forma, os caracteres `'.'` (ponto, representando posição sem peça), `'o'` e `'x'` (representando as peças dos jogadores) podem ser representados pelos grupos de quatro caracteres mostrados abaixo:

```
..    /\    \/
..    \/    /\
```

Escreva o método de forma que uma borda seja colocada entre cada caracter. Uma amostra de como o tabuleiro deve ser impresso é mostrada abaixo (somente as quatro primeiras colunas das duas primeiras linhas são mostradas):

```
+--+--+--+--+
|/\|..|\/|..|
|\/|..|/\|..|
+--+--+--+--+
|/\|..|..|\/|
|\/|..|..|/\|
+--+--+--+--+
```

Dica: Para cada linha do array encapsulado, três linhas devem ser impressas, e, para cada coluna, três caracteres devem ser impressos.

Exercício 11.87: ★★★

Escreva a classe `TabuleiroGenerico` que represente um tabuleiro de jogos como Damas e Xadrez. Escreva, para essa classe, métodos que permitam a inicialização do tabuleiro e a colocação e remoção de peças, assim como a impressão do tabuleiro (com um método `toString`, por exemplo). Modifique a classe `JogoDeDamas` para que esta seja herdeira da classe `TabuleiroGenerico`.

Exercício 11.88: ★★★

Escreva a classe `TabuleiroOitoRainhas` que deve herdar da classe `TabuleiroGenerico` (exercício 11.87) e receber, como argumento para seu construtor, um array de instâncias da classe `Rainha`, e preencher o tabuleiro com as posições das rainhas. *Dica:* Será necessário criar, para a classe `Rainha`, métodos que retornem as coordenadas da rainha no tabuleiro.

Exercício 11.89: ★★★

Escreva, para a classe `TabuleiroOitoRainhas` (exercício 11.88), um método `éSolução` que retorne `true` se a configuração das rainhas passadas como argumento para o construtor da classe for uma solução para o problema das oito rainhas. *Dica:* Use o método `podeSerAtacada` da classe `OitoRainhas` (figura 11.11) como base.

Exercício 11.90: ★★★

Imagine uma solução recursiva para o problema das oito rainhas que tente resolver o problema partindo de uma solução para o problema das sete rainhas (que esperançosamente é mais simples) e tentando posicionar somente a oitava rainha de forma a resolver o problema. O problema das sete rainhas pode ser resolvido usando-se uma solução do problema das seis rainhas e posicionando a sétima adequadamente, e assim por diante. Explique a razão pela qual esse algoritmo, como apresentado, não funcionará, e uma possível correção.

Exercício 11.91: ★★★

Meça o tempo de processamento do algoritmo das oito rainhas (implementado pelas classes `Rainha` e `OitoRainhas`, nas listagens nas figuras 11.10 e 11.11, respectivamente) usando a técnica vista na seção 7.3 (usando o método `currentTimeMillis` da classe `System`). Tente otimizar o algoritmo para que este ache a primeira solução em menos tempo. Alguns pontos que podem ser otimizados são:

- O laço que verifica se uma rainha pode ser atacada por outras pode desistir da busca assim que encontrar uma situação de ataque;
- Quando a rainha é posta no tabuleiro, suas coordenadas iniciais correspondem à primeira posição do tabuleiro, onde certamente poderá ser atacada. Faça com que, quando uma rainha for posta no tabuleiro, ela seja posta numa posição após a penúltima rainha. Assim, se a rainha número 2 estiver no tabuleiro na posição (l=3,c=2) e a rainha número 3 for posta, coloque-a na posição inicial (l=4,c=2) em vez de (l=1,c=1). Adicionalmente, considere colocar a nova rainha uma coluna depois da rainha anterior, para diminuir as possibilidades de ataque.
- Um posicionamento inicial de todas as rainhas em colunas diferentes pode levar a uma solução em menos tempo.

Modifique as classes envolvidas para implementar essas otimizações, uma por vez, verificando o ganho no tempo de execução.

Exercício 11.92: ★★★

Reescreva as classes da solução do problema das oito rainhas de forma que todo o comportamento das oito rainhas seja encapsulado em uma classe `ArrayDeRainhas`. Dessa forma, a classe `OitoRainhas` somente deverá criar uma instância da classe `ArrayDeRainhas` e executar o método `soluciona` desta para tentar solucionar o problema.

Exercício 11.93: ★★★

Os elementos do array encapsulado pela classe `ArrayDeObjetosGeometricos` (figura 11.7) são cópias das referências criadas pela classe que usar instâncias de `ArrayDeObjetosGeometricos`, e podem ser modificadas depois de inseridas. Uma solução para evitar isso é garantir que os objetos inseridos no array são clones dos objetos passados como argumento para o método `modifica`. Modifique o método `modifica` para que este use clones dos objetos quando for inseri-los no array. Veja também o exercício 9.24.

Capítulo 12

Classes para manipulação de strings

12.1 Exercícios do Capítulo 12

Os exercícios deste capítulo que envolvem conceitos de biologia computacional são, em geral, versões simplificadas de algoritmos e problemas reais.

Alguns dos algoritmos descritos nesta seção (e suas variações) podem ser conhecidos por outros nomes.

Exercício 12.1: ⋆

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

```
1  class DemoString
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 12.2: ⋆

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 12.3: ⋆

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 12.4: ★

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 12.5: ★

Escreva, para a classe `URL` (figura 12.5), um método `éHTML` que retorne `true` caso a URL termine com `"htm"` ou `"html"`.

Exercício 12.6: ★

Escreva, para a classe `URL` (figura 12.5), um método `éFilme` que retorne `true` caso a URL termine com `"mov"`, `"avi"`, `"rm"` ou `"mpeg"`.

Exercício 12.7: ★

Modifique a classe `JogoDaForca` (figura 12.1) 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 12.8: ★★

Crie, na classe `StringUtils` (figura 12.8), 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 12.9: ★★

Crie, na classe `StringUtils`, 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 12.10: ★★

Crie, na classe `StringUtils`, 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 12.11: ★★

Escreva na classe `StringUtils` (figura 12.8) 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 12.12: ★★

Modifique a classe `JogoDaForca` (figura 12.1) 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 12.8).

Exercício 12.13: ★★

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 12.14: ★★

Escreva na classe `StringUtils` vários métodos sobrecarregados (veja seção 4.3) 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 12.15: ★★

Escreva na classe `StringUtils` (veja exercício 12.8) 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`.

Exercício 12.16: ★★

Modifique a classe `URL` (figura 12.5) para que todas as combinações possíveis de maiúsculas e minúsculas sejam consideradas nos métodos `éImagem` e `éHTTP`. *Dica:* Mantenha uma versão da URL encapsulada somente com caracteres minúsculos, eliminando a necessidade de comparação com maiúsculos.

Exercício 12.17: ★★

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 12.18: ★★

Escreva um método `listaTerminais` na classe `StringUtils` (figura 12.8) que receba uma string e um caracter como argumento, e imprima todas as substrings da string passada que terminem com o caracter passado. Por exemplo, se a string `"indeterminadamente"` e o caracter `'e'` forem passados como argumentos, o método deverá imprimir as strings `"inde"`, `"indete"`, `"indeterminadame"`, `"indeterminadamente"`.

Exercício 12.19: ★★

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 12.20: ★★

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 12.21: ★★

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 12.22: ★★★

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 `"Javanes"` e `"Japones"` têm o mesmo código (J152), a string `"assimétrico"` é codificada como A253 e a string `"Java"` é codificada como J100.
Crie, na classe `StringUtils` (figura 12.8), um método estático que implemente o algoritmo *soundex*.

Exercício 12.23: ★★★

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). Escreva dois métodos estáticos na classe `StringUtils` (figura 12.8) que retornem `true` se uma string passada como argumento for palíndroma e `false` se não for. Um dos métodos deve ser estrito e considerar espaços como caracteres, o outro não – como diferenciar os dois? *Dica:* Use o exercício 12.15 como base.

Exercício 12.24: ★★★

Escreva uma classe `ArrayDeStrings`, no molde da classe `ArrayDeFloats` (figura 11.4) que encapsule um array de strings. Essa classe deve ter os seguintes métodos:

- Construtor, que recebe como argumento um número máximo de instâncias da classe `String` que serão encapsuladas pela classe,
- `tamanho`, que retorna o número de strings no array encapsulado,
- `modifica`, que recebe como argumentos um valor inteiro (posição no array) e uma string, e faz com que a string naquela posição do array passe a ser a passada como argumento,
- `valor`, que recebe como argumento um valor inteiro (posição) e retorna a string armazenada naquela posição do array,
- `toString`, que retorna uma única string contendo todas as strings encapsuladas na classe, separadas por quebra de linhas (caracter `'\n'`).

Exercício 12.25: ★★★

Escreva, para a classe `ArrayDeStrings`, um método `existe` que receba como argumento uma string e retorne `true` caso a string passada como argumento exista (exatamente igual) no array de strings encapsulado pela classe. Escreva também um método `existeIgnoreCase` que execute a mesma tarefa mas considerando que caracteres maiúsculos e minúsculos são iguais.

Exercício 12.26: ★★★

Escreva um método estático `removeCaracteres` na classe `StringUtils` que, recebendo duas strings como argumentos, retorne uma nova string que é a subtração dos caracteres da segunda string passada como argumento dos caracteres da primeira string. Por exemplo, se as strings `"Abracadabra"` e `"cabra"` forem passadas como argumentos, o algoritmo deve remover **uma** de cada uma das letras de `"cabra"` da palavra `"Abracadabra"`, resultando em `"Adabra"`. Se algum caracter da segunda string não existir na primeira, o método deve retornar uma string vazia (por exemplo: se a segunda string for `"praxe"` e a primeira for `"paralelepípedo"`, a subtração não poderá ser efetuada).

Exercício 12.27: ★★★

Um dos problemas com instâncias da classe `StringTokenizer` é que não podemos verificar que *tokens* já foram extraídos – só existem mecanismos para recuperar o próximo *token*, mas não para recuperar *tokens* anteriores. Crie uma classe `StringTokenizerComoArray` que encapsule um array de strings que seja obtido através de uma instância da classe `StringTokenizer`. O construtor dessa classe deverá alocar e inicializar o array com os *tokens* de uma string passada como argumento, e a classe deve ter métodos que permitam a obtenção de qualquer *token*. *Dica:* Veja a classe `ArrayDeFloats` (figura 11.4) e o exercício 12.24.

Exercício 12.28: ★★★

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 12.29: ★★★

Escreva, para a classe `StringDNA` (exercício 12.28), 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 12.30: ★★★

Crie, na classe `StringDNA` (veja exercício 12.28), 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 12.31: ★★★

Uma string contendo RNA é composta somente dos caracteres `'A'`, `'C'`, `'G'` e `'U'` – nenhum outro caracter é permitido. Escreva uma classe `StringRNA` que herde da classe `StringDNA` (exercício 12.28) e implemente os mesmos métodos da classe `StringDNA`. Quais métodos devem ser sobrescritos?

Exercício 12.32: ★★★

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 12.33: ★★★

O *algoritmo de César* de criptografia de strings é uma versão melhorada do algoritmo *rot13* (veja exercício 12.32): 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 12.34: ★★★

O mecanismo de casamento de padrões mostrado na classe `CasamentoDePadroes` (figura 12.10) considera caracteres maiúsculos e minúsculos como sendo diferentes. Modifique a classe `CasamentoDePadroes` de forma que esta encapsule também um valor booleano `ignoreCase`, que deve ser inicializado pelo construtor, usando um argumento. Se esse valor booleano for igual a `true`, o método `casa` deverá considerar caracteres maiúsculos iguais a minúsculos. *Dica:* A resposta do exercício 12.21 poderá ser de ajuda.

Exercício 12.35: ★★★

Escreva, para a classe `StringUtils` (figura 12.8) o método `éSubSeqüênciaDe` que recebe duas strings como argumentos e retorna `true` se a primeira string for subseqüência da segunda, e `false` se não for. Uma string é considerada uma subseqüência de outra se todos os caracteres da primeira string aparecerem em ordem na segunda. Por exemplo, considerando a string `palavra` valendo `"consciensiosamente"`, a string `"ciente"` seria subseqüência de `palavra` enquando `"cimento"` não seria. Podemos também definir que uma string `a` é uma subseqüência de outra string `b` se pudermos formar `a` retirando caracteres de `b` mas mantendo a ordem.

Exercício 12.36: ★★★

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 12.37: ★★★

Modifique o construtor da classe `CasamentoDePadroes` (figura 12.10) de forma que este verifique se o padrão pode ser analisado sem problemas. O padrão deve ter, para cada caracter `'{'`, um caracter `'}'` correspondente, e não deve ter pares de caracteres `'{'` e `'}'` aninhados.

Exercício 12.38: ★★★

Escreva um método na classe `CasamentoDePadroes` que calcule e retorne quantas strings diferentes podem ser formadas com o padrão encapsulado. Como referência, considere que o padrão `"{1,2,3,4}/{8,9,10}/2001"` pode formar 12 strings diferentes.

Exercício 12.39: ★★★

Usando o exercício 12.38 como base, escreva um método que imprima todas as strings diferentes que podem ser formadas com o padrão encapsulado.

Exercício 12.40: ★★★

Escreva, para a classe `StringUtils`, o método `éSubSetDe` que recebe duas strings como argumentos e retorna `true` se a primeira string for subset da segunda, e `false` se não for. Uma string é considerada um subset de outra se todos os caracteres da primeira string aparecerem em qualquer ordem na segunda, mas sem repetir caracteres da segunda. Por exemplo, considerando a string `palavra` valendo `"consciensiosamente"`, as string `"ciente"` e `"cimento"` seriam subsets de `palavra`, mas `"consomem"` não seria pois requer dois caracteres `'m'`, e a string `palavra` somente contém um.

Exercício 12.41: ★★★

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 12.42: ★★★

Escreva, para a classe `StringUtils` (figura 12.8), 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`" poderia ser "`desncotram`", e o alfabeto completo de "`colina`" poderia ser a própria string "`colina`".

Exercício 12.43: ★★★

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 12.44: ★★★★

Escreva, para a classe `StringUtils` (figura 12.8), 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 12.23.

Exercício 12.45: ★★★★

Escreva a classe `CasamentoDePadroesComCoringa` que execute o algoritmo de casamento de padrões mostrado na seção 12.4 mas considerando que o caracter `'?'` também possa ser usado, e que o algoritmo de casamento considere que esse caracter é igual a qualquer outro caracter que apareça na string sendo analisada. Dessa forma, se o padrão encapsulado for `"{re,di}??de"` as strings `"reside"`, `"divide"` e `"revide"` casarão com o padrão, enquanto as strings `"rebelde"` e `"relate"` não casarão. *Dicas:* Considere as diferenças do algoritmo no tratamento de caracteres `'?'` como sendo um *singleton* e como sendo parte da lista de alternativas. Vale a pena escrever essa classe como sendo herdeira da classe `CasamentoDePadroes`? Quais as vantagens e dificuldades esperadas?

Exercício 12.46: ★★★★

Duas seqüências 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 seqüências, mas freqüentemente 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 seqüências 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 12.49) um método `calculaAliamento` que receba como argumento outra instância da classe `StringAminoAcidos` e retorne o valor do alinhamento da string encapsulada com a passada como argumento.

Exercício 12.47: ★★★★

Escreva, para a classe `ArrayDeStrings`, um método `ordenaCrescente` que ordene as strings contidas no array encapsulado em ordem crescente (alfabética) usando o algoritmo *bubblesort* e retorne o array ordenado. Escreva também um método `ordenaDecrescente` que ordene as strings em ordem decrescente. *Dica:* Veja o exercício 11.70, e use o método `compareTo` da classe `String` para comparar as strings para determinar qual vem antes em ordem crescente.

Exercício 12.48: ★★★★

Escreva, para a classe `ArrayDeStrings` (exercício 12.24), um método `união` que receba, como argumento, uma outra instância da classe `ArrayDeStrings` e retorne uma nova instância da classe contendo a união das strings do array encapsulado com as do passado como argumento, sem repetições. Por exemplo, se o array encapsulado contiver as strings [`"célula"`, `"núcleo"`, `"plasma"`, `"mitocôndria"`] e o array passado como argumento para o método contiver [`"átomo"`, `"núcleo"`, `"plasma"`, `"molécula"`], o método deverá retornar o array [`"célula"`, `"núcleo"`, `"plasma"`, `"mitocôndria"`, `"átomo"`, `"molécula"`]. Escreva também o método `uniãoIgnoreCase`, que considera caracteres maiúsculos como sendo iguais aos minúsculos.

Exercício 12.49: ★★★★

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`

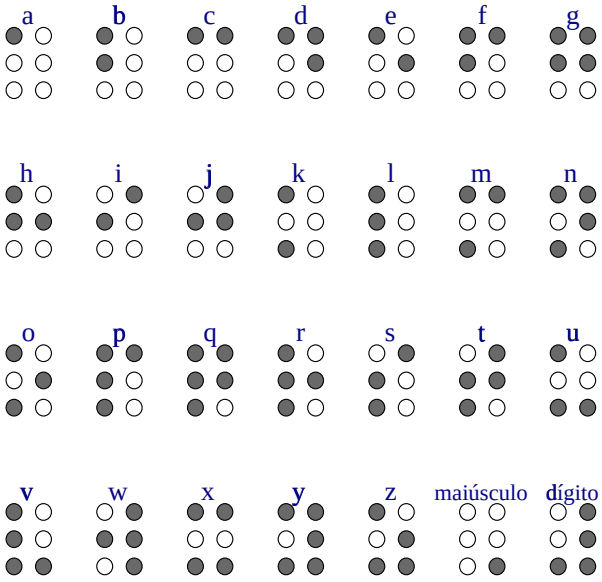
Considerando a lista acima, escreva a classe `StringAminoAcidos`, que encapsule uma string composta somente de símbolos de aminoácidos. O construtor dessa classe deve receber como argumento uma instância da classe `StringRNA` (exercício 12.31) e transformar grupos de três em três caracteres para símbolos 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 12.50: ★★★★

O *algoritmo de César* (veja o exercício 12.33) 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 `codificaCésar` na classe `Criptografia` (veja exercício 12.32) 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 `decodificaCésar`, 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 12.51: ★★★★

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 12.52: ★★★★

Várias seqüências 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 seqüências de caracteres

```
''FNTXSPRNCDE''
''FCXTSRNRPDE''
''NNTXSRPNCCE''
''FNTXSPXRNDE''
```

o consenso seria calculado para cada posição como sendo o caracter mais freqüente, e o resultado seria a string `"FNTXS???CDE"`. Como em três posições não houve consenso (houve empate entre os caracteres mais freqüentes), consideramos o caracter na posição como sendo igual a `'?'`.
Escreva na classe `StringAminoAcidos` (exercício 12.49) 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 12.53: ★★★★

Escreva, para a classe `ArrayDeStrings` (exercício 12.24), um método `interseção` que receba, como argumento, uma outra instância da classe `ArrayDeStrings` e retorne uma nova instância da classe contendo a interseção das strings do array encapsulado com as do passado como argumento. Por exemplo, se o array encapsulado contiver as strings [“célula”, “núcleo”, “plasma”, “mitocôndria”] e o array passado como argumento para o método contiver [“átomo”, “núcleo”, “plasma”, “molécula”], o método deverá retornar o array [“núcleo”, “plasma”]. Escreva também o método `interseçãoIgnoreCase`, que considera caracteres maiúsculos como sendo iguais aos minúsculos.

Exercício 12.54: ★★★★

Escreva, para a classe `StringBraille` (exercício 12.51), 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 12.51.

Exercício 12.55: ★★★★

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` (figura 12.8), um método `éAnagrama` que receba duas strings como argumentos e retorne `true` se uma é anagrama de outra.

Exercício 12.56: ★★★★★

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 12.57: ★★★★★

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
```

12.2 Exercícios complementares do Capítulo 12

Exercício 12.58: ★

A forma pela qual o jogo da forca é implementada pela classe `DemoJogoDaForca` (figura 12.2) permite que o jogador tente até acertar a palavra. Modifique o método `main` da classe `DemoJogoDaForca` para que somente um determinado número de tentativas possa ser feito. Esse número pode ser calculado como $2c/3$, onde c é o comprimento da palavra.

Exercício 12.59: ★★

O método `pergunta` da classe `JogoDaForca` (figura 12.1) não verifica se o usuário entrou uma letra das que já foram tentadas anteriormente. Modifique esse método para que a pergunta seja feita enquanto o usuário não entrar uma letra que não tenha sido utilizada anteriormente.

Exercício 12.60: ★★

Modifique a classe `EpocaDeFrutas` (figura 12.7) para que vários nomes de meses possam ser passados como argumentos pela linha de comando, e que o programa imprima todas as frutas que podem ser colhidas em qualquer um dos meses passados como argumentos.

Exercício 12.61: ★★

Escreva um construtor para a classe `ArrayDeFloats` (figura 11.4) que receba como argumento uma única string contendo vários valores de ponto flutuante separados por espaços, e que inicialize os elementos do array com esses valores, com o processamento adequado.

Exercício 12.62: ★★

Considere o construtor da classe `RegistroAcademico` (exercício 6.22). Faça uma outra versão desse construtor onde a informação sobre o curso é passada como uma string contendo o nome do curso. O construtor deverá, a partir da string passada como argumento para identificar o curso, reconhecer o código correspondente àquele curso. *Dica:* Tente escrever um método versátil, que reconheça tanto "`Engenharia da Computação`" quanto "`engenharia da computação`" quanto " `engenharia da computação` " como identificadores correspondentes ao código 39.

Exercício 12.63: ★★

Modifique o resultado do exercício 12.60 para que a classe verifique se os parâmetros `-help` ou `-h` foram passados pela linha de comando e, se tiverem sido passados, que imprima uma breve mensagem sobre o uso do programa, saindo em seguida.

Exercício 12.64: ★★

Existem vários pontos na classe `CasamentoDePadroes` (figura 12.10) que podem ser otimizados para que o algoritmo de casamento seja mais eficiente. Um dos pontos é a eliminação da necessidade de recriar a instância da classe `StringTokenizer` para os tokens do padrão a cada chamada do método `casa` (veja os exercícios 12.27 e 12.38 para idéias de como eliminar essa necessidade). Você consegue encontrar outros pontos para otimização?

Exercício 12.65: ★★★

Considere a classe `DemoLogin` (figura 12.4), que verifica que nome de autor foi entrado e imprime uma frase desse autor. Escreva a classe `Citacoes` que encapsule, usando arrays de strings, autores e respectivas citações. Escreva nessa classe um método `menu` que mostre, em um menu, os nomes dos autores, permitindo ao usuário escolher um nome e ver a(s) frase(s) desse autor. Todos os métodos dessa classe podem ser estáticos.

Exercício 12.66: ★★★

Escreva uma versão recursiva do método `replica` (exercício 12.10).

Exercício 12.67: ★★★

Escreva o método `reverte` da classe `StringUtils` (pedido no exercício 12.15) de forma recursiva. Um método recursivo que reverta uma string pode ser escrito com os seguintes passos:

- Se a string a ser revertida for vazia, retorne uma string vazia;
- Senão, retorne o último caracter da string concatenado com o resultado da chamada do método, passando como argumento a string sem o último caracter.

Existe uma maneira simples de otimizar esse algoritmo, tente achá-la.

Exercício 12.68: ★★★

Escreva uma versão recursiva do método `quantasVezes` (exercício 12.19).

Exercício 12.69: ★★★

Escreva uma versão recursiva do método estrito (que considera espaços) que verifica se uma string é palíndroma (exercício 12.23). Esse método pode ter os seguintes passos:

- Se a string a ser revertida for vazia ou tiver somente um único caracter, retorna `true`.
- Senão, verifica o primeiro e último caracteres da string. Se forem iguais, retorna o valor do método, passando como argumento para este a string sem o primeiro e últimos caracteres. Se o primeiro e último caracteres forem diferentes, retorna `false`.

Capítulo 13

Coleções de objetos

13.1 Exercícios do Capítulo 13

Exercício 13.1: ⋆

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

```
1  import java.util.*;
2  class TestaLista
3    {
4    public static void main(String[] argumentos)
5      {
6      List aa = new LinkedList();
7      aa.add("Alanina"); aa.add("Leucina"); aa.add("Alanina"); aa.add("Triptofano");
8      List aa2 = new LinkedList();
9      aa2.add("Alanina"); aa2.add("Leucina"); aa2.add("Alanina"); aa2.add("Alanina");
10     }
11   }
```

A. O resultado de `aa.containsAll(aa2)` será `true`.
B. O resultado de `aa2.containsAll(aa)` será `true`.
C. O resultado de `aa.containsAll(aa)` será `true`.

Exercício 13.2: ⋆

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

```
1  import java.util.*;
2  class TestaLista
3    {
4    public static void main(String[] argumentos)
5      {
6      List a = new LinkedList();
7      a.add("um"); a.add("dois"); a.add("três"); a.add("quatro"); a.add("um");
8      List b = new LinkedList();
9      b.add("um"); b.add("um");   b.add("três");
10     a.retainAll(b);
11     }
12   }
```

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 13.3: ★

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

```
1  import java.util.*;
2  class TestaLista
3    {
4    public static void main(String[] argumentos)
5      {
6      List a = new LinkedList();
7      a.add("um"); a.add("dois"); a.add("três"); a.add("quatro"); a.add("um");
8      List b = new LinkedList();
9      b.add("um"); b.add("um");   b.add("três");
10     a.removeAll(b);
11     }
12   }
```

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 13.4: ★

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

Exercício 13.5: ★

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

Exercício 13.6: ★

Escreva um método para a classe `ListaDePedidos` (figura 13.16) que retorne o número total de pedidos na fila de prioridades.

Exercício 13.7: ★

No método `main` da classe `DemoAvaliadorDeExpressoes` (figura 13.20), a expressão "`9 5 1 +`" foi avaliada incorretamente como sendo igual a 6. Explique.

Exercício 13.8: ★★

Usando a classe `OperacoesComConjuntos` (figura 13.2) como base, calcule e imprima o conjunto de todos os autores não-atletas.

Exercício 13.9: ★★

Usando a classe `OperacoesComConjuntos` (figura 13.2) como base, calcule e imprima o conjunto de todos os autores que são nadadores **ou** casados.

Exercício 13.10: ★★

Escreva, para a classe `ListaDePalavras` (figura 13.3), um método `toString` que retorne as palavras da lista encapsulada, concatenadas e separadas por espaços.

Exercício 13.11: ★★

Escreva, para a classe `ListaDePalavras` (figura 13.3), um método `remove` que receba uma string como argumento e que remova este argumento da lista de palavras encapsuladas.

Exercício 13.12: ★★

Escreva para a classe `ListaDePalavras` (figura 13.3) 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 de palavras.

Exercício 13.13: ★★

Escreva para a classe `ListaDePalavras` (figura 13.3) 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 13.12.

Exercício 13.14: ★★

Escreva para a classe `ListaDePalavras` (figura 13.3) uma versão sobrecarregada do método `contémTodas` que receba como argumento uma outra instância da classe `ListaDePalavras` e que retorne `true` se cada uma dos elementos do array existir na lista encapsulada de palavras. Veja também os exercícios 13.12 e 13.13. *Dica:* A solução desse exercício pode ser implementada com operações sobre os conjuntos.

Exercício 13.15: ★★

Escreva, para a classe `ListaDePalavras` (figura 13.3), um método `removeTodas` que receba um array de strings como argumento e que remova todas as strings do array da lista de palavras encapsuladas.

Exercício 13.16: ★★

Escreva, para a classe `ListaDePalavras` (figura 13.3), um método `removeTodas` que receba como argumento outra instância da classe `ListaDePalavras` e que remova todas as palavras da lista passada como argumento da lista de palavras encapsuladas. Veja também o exercício 13.15. *Dica:* A solução desse exercício pode ser implementada com operações sobre os conjuntos.

Exercício 13.17: ★★

Escreva, para a classe `ListaDePalavras` (figura 13.3), um método `união` que receba como argumento uma outra instância de `ListaDePalavras` e que retorne uma nova instância de `ListaDePalavras` contendo a união da lista de palavras encapsulada com a passada como argumento.

Exercício 13.18: ★★

Escreva, para a classe `ListaDePalavras` (figura 13.3), um método `interseção` que receba como argumento uma outra instância de `ListaDePalavras` e que retorne uma nova instância de `ListaDePalavras` contendo a interseção da lista de palavras encapsulada com a passada como argumento.

Exercício 13.19: ★★

Escreva e demonstre uma rotina 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 13.20: ★★

Escreva um método para a classe `ListaDePedidos` (figura 13.16) que retorne uma lista com os dez pedidos mais prioritários. Se na fila de pedidos houver menos de dez pedidos no total, estes devem ser retornados.

Exercício 13.21: ★★

Alguns sites na Internet têm mais de uma máquina respondendo por eles, com mais de um número IP correspondendo ao nome do site. Reescreva a classe `DicionarioDeIPs` (figura 13.21) de forma que seja possível armazenar e recuperar arrays de IPs associados a nomes de sites. Uma maneira é associar a cada nome de site uma lista de IPs.

Exercício 13.22: ★★

Modifique a classe `DicionarioDeIPs` (figura 13.21) para que a procura possa ser feita pelo IP e pelo nome do site, sem alterar a estrutura do mapa.

Exercício 13.23: ★★

Escreva, para a classe `ArrayEsparsoDeDoubles` (figura 13.22), 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 13.24: ★★

Escreva, para a classe `ArrayEsparsoDeDoubles` (figura 13.22), um método `total` que calcule e retorne o total dos valores presentes no array.

Exercício 13.25: ★★

Escreva, para a classe `ArrayEsparsoDeDoubles`, um método `união` que receba como argumento uma outra instância de `ArrayEsparsoDeDoubles` e que retorne uma nova instância da mesma classe contendo a união do array esparso encapsulado com o passado como argumento.

Exercício 13.26: ★★

Escreva, para a classe `ArrayEsparsoDeDoubles`, um método `interseção` que receba como argumento uma outra instância de `ArrayEsparsoDeDoubles` e que retorne uma nova instância da mesma classe contendo a interseção do array esparso encapsulado com o passado como argumento.

Exercício 13.27: ★★

Escreva para a classe `ArrayEsparsoDeDoubles` (figura 13.22) um método `menor` que retorne o menor valor dentro do array esparso.

Exercício 13.28: ★★

Escreva para a classe `ArrayEsparsoDeDoubles` (figura 13.22) um método `maior` que retorne o maior valor dentro do array esparso.

Exercício 13.29: ★★

Usando a classe `ListaDeNomes` (figura 13.27) como base, demonstre os métodos `max` e `min` da classe `Collections`.

Exercício 13.30: ★★★

Modifique o método `remove` da classe `MaquinaDeKaraoke` (figura 13.10) para que este remova a última ocorrência da música passada como argumento, em vez da primeira.

Exercício 13.31: ★★★

Escreva para a classe `MaquinaDeKaraoke` (figura 13.10) um método `removeTodas` que remova todas as ocorrências da música passada como argumento.

Exercício 13.32: ★★★

Escreva para a classe `MaquinaDeKaraoke` (figura 13.10) dois métodos, `adia` e `adianta`, que recebam um nome de música como argumento e que, respectivamente, adiaem ou adiantem a execução daquela música, 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 13.33: ★★★

Modifique o método `entraPedido` da classe `ListaDePedidos` (figura 13.16) para que este verifique se um pedido do mesmo material já foi feito antes de adicionar. Se na fila de prioridades já houver um pedido do mesmo material, a quantidade deve ser adicionada ao pedido já feito. Consideremos como exemplo os pedidos do material "Anestésico X-45" do método `main` da classe `DemoListaDePedidos` (figura 13.17: o segundo pedido não deveria ser lançado na fila de pedidos, e o primeiro pedido deveria ter sua quantidade modificada de 20 para 120. *Dica:* Para a solução deste exercício será necessário modificar a classe `Pedido` (figura 13.13).

Exercício 13.34: ★★★

Escreva, para a classe `ListaDePedidos` (figura 13.16), dois métodos que aumentem ou diminuam a prioridade de determinado pedido. Esses métodos devem receber como argumento uma instância de qualquer classe que herde da classe `Pedido` e devem incrementar ou decrementar a prioridade daquele pedido.

Exercício 13.35: ★★★

Escreva uma classe `ContadorDePalavras` baseada na classe `ListaDePalavras` 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 13.36: ★★★

Usando as classes que demonstram a ordenação customizada de uma turma de alunos (figuras 13.28 a 13.32) escreva e modifique classes que listem os alunos ordenados pela idade (menor para maior).

Exercício 13.37: ★★★

Usando as classes que demonstram a ordenação customizada de uma turma de alunos (figuras 13.28 a 13.32) escreva e modifique classes que listem os alunos ordenados pelo número de disciplinas que já foram cursadas (mais disciplinas para menos disciplinas).

Exercício 13.38: ★★★★

Escreva, para a classe `AvaliadorDeExpressoes` (figura 13.19), um método estático `converteInfixParaPostfix` que converta uma string contendo uma expressão *infix* (onde operadores são escritos entre operandos e parênteses são usados para agrupar operações) para *postfix*, para que esta possa ser processada pelo método `avaliaExpressãoPostfix` da mesma classe. O algoritmo de conversão de *infix* para *postfix* é mostrado a seguir:

1. Usamos como entrada para o algoritmo uma lista de operadores e operandos que possa ser quebrada em *tokens* e como saída uma string que conterá o resultado da conversão. Cada *token* será processado pelos passos seguintes:
2. Se o *token* for um valor numérico, este será concatenado na string de conversão.
3. Se o *token* for um parêntese esquerdo, será ignorado.
4. Se o *token* for um operador, ele deverá ser armazenado no topo de uma pilha de operadores.
5. Se o *token* for um parêntese direito, o operador no topo da pilha de operadores deverá ser concatenado à string de conversão.

Para auxiliar na resolução deste exercício, crie uma classe `PilhaDeOperadores` para conter os operadores usados na conversão. Crie também na classe `AvaliadorDeExpressoes` um método estático `avaliaExpressãoInfix` que avalie expressões do tipo *infix* usando o método `converteInfixParaPostfix`.

Exercício 13.39: ★★★★

Considere os exercícios 13.27 e 13.27. Reescreva a classe `ArrayEsparsoDeDoubles` para que a busca do maior e menor valor seja feita usando os métodos `min` e `max` da classe `Collections`. *Dicas:* Existem duas soluções para este problema: uma (a mais simples) é obter a lista de valores do mapa encapsulado pela classe `ArrayEsparsoDeDoubles` e usar os métodos `max` e `min` diretamente. A segunda solução (mais complexa) é reescrever `ArrayEsparsoDeDoubles` para usar uma lista de pares índice-valor, que devem ser comparáveis (implementando `Comparable`).

Exercício 13.40: ★★★★

Usando a classe `ArrayDeObjetosGeometricos` como base, crie classes que representam uma lista de objetos geométricos. Escreva, para essa classe, métodos que mostrem a lista de objetos ordenados pelas suas áreas. *Dica:* Será necessário modificar várias classes apresentadas anteriormente.

Exercício 13.41: ★★★★★

Escreva uma classe que implemente um comparador imperfeito para strings, isto é, um comparador que eventualmente erre no resultado da comparação. Isso pode ser feito usando-se o método `Math.random`, que retorna um valor aleatório entre zero e um, para interferir no resultado da comparação. O método `compare` dessa classe pode comparar o resultado de `Math.random` com uma constante (por exemplo, 0.9), e, se o valor aleatório for maior que essa constante, o resultado da comparação deverá ser falsificado de alguma forma. Com essa classe, demonstre o método `sort` para ordenar uma lista de strings. Discuta e justifique os resultados.

13.2 Exercícios complementares do Capítulo 13

Exercício 13.42: ⋆

Modifique o método `entraPedido` da classe `ListaDePedidos` (figura 13.16) para que este aceite prioridades fora da faixa de valores válidos (0 a 9). Prioridades menores do que zero devem ser consideradas como zero e prioridades maiores do que nove devem ser consideradas iguais a nove.

Exercício 13.43: ⋆⋆

Modifique a classe `RoboComMemoria` (figura 13.6) para que o robô também tenha uma bateria cuja capacidade diminui a cada movimento do robô (veja também a figura 9.3).

Exercício 13.44: ⋆⋆

Escreva um método para a classe criada no exercício 13.43 que retorne qual será a energia necessária para o robô retornar à base.

Exercício 13.45: ⋆⋆

Os métodos da classe `RoboComMemoria` (figura 13.6) não consideram a primeira posição do robô – podemos ver isso no resultado da execução da classe `DemoRoboComMemoria` (figura 13.7). Modifique os métodos necessários para que, ao executar o método `retornaÀBase`, o robô esteja na posição original **e** na direção original.

Exercício 13.46: ⋆⋆

Modifique a classe `ListaDePalavras` (figura 13.3) para que somente as palavras sejam armazenadas na lista (isto é, evitando armazenar caracteres de pontuação) e para que palavras sejam armazenadas e verificadas na lista independentemente de estarem em maiúsculas ou minúsculas. Com essa modificação, e usando a listagem na figura 13.4, uma chamada ao método `palavrasDoMillôr.existe("DEMOCRACIA")` deverá retornar `true`. *Dica:* Podemos modificar a classe para que somente uma forma das palavras seja armazenada – por exemplo, podemos armazenar somente as palavras em minúsculas.

Exercício 13.47: ⋆⋆

Modifique o método `entraPedido` da classe `ListaDePedidos` (figura 13.16) para que este não aceite prioridades fora da faixa de valores válidos (0 a 9). Se alguma prioridade inválida for passada, o método deverá lançar a exceção `IllegalArgumentException`. Veja também o exercício 13.42.

Exercício 13.48: ⋆⋆⋆

O algoritmo de avaliação de expressões mostrado no método `avaliaExpressãoPostfix` da classe `AvaliadorDeExpressoes` (figura 13.19) tem algumas falhas, conforme pode ser visto no método `main` da classe `DemoAvaliadorDeExpressoes` (figura 13.20): o algoritmo não reconhece números precedidos de um sinal. Modifique o algoritmo para que o mesmo saiba diferenciar entre um operador (que deve ser um único caracter) e um operando que possa ter opcionalmente um sinal `'+'` ou `'-'` precedendo-o.

Bibliografia comentada

Dois livros introdutórios interessantes sobre Java são *An Introduction to Computer Science Using Java* de Samuel N. Kamin, M. Dennis Mickunas e Edward M. Reingold (ISBN 0-07-034224-5, McGraw-Hill, 1998) e *Introduction to Programming Using Java – An Object-Oriented Approach* de David Arnow e Gerald Weiss (ISBN 0-201-75159-3, Addison-Wesley, 1998). Estes livros podem ser usados como leitura complementar a este.

Alguns tutoriais *on-line* são recomendados para os interessados na linguagem Java: o tutorial da Sun cobre vários aspectos da linguagem Java, desde introdução e aspectos práticos até tópicos intermediários como *applets* e interfaces gráficas. Este tutorial pode ser acessado *on-line* em http://java.sun.com/docs/books/tutorial/ ou copiado para seu computador. Alguns tutoriais da IBM também podem ser de interesse do leitor: o site *developerWorks* da IBM (http://www-106.ibm.com/developerworks/) contêm vários tutoriais sobre Java e desenvolvimento em geral. Em particular, os tutoriais *Java language essentials* e *Java collections framework* são interessantes para estudo posterior.

Para uma visão mais completa de Java, recomendo os livros *Core Java 2 – Fundamentals* e *Core Java 2 – Advanced Features*, de Cay S. Horstmann e Gary Cornell (ISBN 0-13-089468-0 e 0-13-081934-4, Prentice Hall, 2001). Estes livros foram traduzidos para o português pela editora Makron, com os títulos *Core Java 2 – Volume 1 – Fundamentos* e *Core Java 2 – Volume 2 – Recursos Avançados* (ISBN 85-346-1225-0 e 85-346-1253-6).

Alguns dos exercícios[1] foram inspirados pelo livro *Complete Java 2 Certification Study Guide*, de Simon Roberts, Philip Heller e Michael Ernest (ISBN 0-7821-4077-7, Sybex, 2000) – livro recomendado para quem já tem ao menos noções de Java e quer estudar para uma das certificações da Sun.

Três livros descrevem, com detalhes, as classes padrão de Java e seus métodos e campos: *The Java Class Libraries, Second Edition, Volume One*, de Patrick Chan, Rosanna Lee e Douglas Kramer (ISBN 0-201-31002-3); *The Java Class Libraries, Second Edition, Volume Two*, de Patrick Chan e Rosanna Lee (ISBN 0-201-31003-1); e *The Java Class Libraries, Second Edition, Volume One, Supplement for the Java 2 Platform, Standard Edition, v1.2*, de Patrick Chan, Rosanna Lee e Douglas Kramer (ISBN 0-201-48552-4). Os três livros foram publicados pela Addison-Wesley. Os dois volumes originais são de 1998 e o complemento de 1999. Apesar de cobrirem somente a versão 1.2 do Java, contêm muitos exemplos de uso dos diversos métodos, e são uma ótima referência para o programador intermediário.

Vários livros foram usados como inspiração para os exercícios propostos que envolvem algoritmos, otimização, recursividade e coleções. Desses, destaco *Data Structures and Algorithms in Java*, de Michael T. Goodrich e Roberto Tamassia (ISBN 0-471-38367-8, John Wiley and Sons, Inc., 1998, recentemente traduzido para o português com o título *Estruturas de Dados e Algoritmos em Java*, publicado pela Bookman Companhia Editorial); *Algorithms in C++*, de Robert Sedgewick (ISBN 0-201-51059-6, Addison-Wesley, 1992) e *Algorithmics – The Spirit of Computing, second edition* de David Harel (ISBN 0-201-50401-4, Addison-Wesley, 1992).

Outros dois livros interessantes sobre algoritmos e técnicas de programação são *Introduction to Computer Science – An Algorithmic Approach*, de Jean-Paul Tremblay e Richard B. Bunt (ISBN 0-07-065167-1, McGraw-Hill, 1979) e *A Collection of Programming Problems and Techniques*, de H. A. Maurer e M. R. Williams (ISBN 0-13-139592-0, Prentice-Hall, 1972). Vários dos exercícios foram inspirados por esses dois livros.

O livro *Cryptography: Theory and Practice*, de Douglas R. Stinson (ISBN 0-8493-8521-0, CRC Press, 1995) foi usado para revisão dos algoritmos básicos de criptografia usados em alguns dos exercícios.

Todos os exercícios sobre séries matemáticas no Capítulo 7 foram retirados do livro *Manual de Fórmulas e Tabelas Matemáticas*, de Murray R. Spiegel (ISBN 0-07-090032-9, coleção Schaum da editora McGraw-Hill, 1973), da página "Mathematical Constants and Computation", de Xavier Gourdon e Pascal Sebah (http://numbers.computation.free.fr/Constants/constants.html), do artigo "On the rapid computation of various polylogarithmic constants" de David Bailey, Peter Borwein e Simon Plouffe e da página da Mathsoft (http://www.mathsoft.com). Estas referências contêm muitas outras fórmulas sobre séries matemáticas que não foram aproveitadas neste livro.

[1] Os exercícios são apresentados em um material complementar que pode ser copiado da Internet, veja o prefácio deste livro. Referências para os exercícios são apresentadas aqui para tornar esta lista de referências mais completa.

Alguns dos exercícios sobre biologia computacional foram inspirados pelos livros *Introduction to Computational Molecular Biology* de João Carlos Setúbal e João Meidanis (ISBN 0-534-95262-3, PWS Publishing Company, 1997) e *Algorithms on Strings, Trees and Sequences*, de Dan Gusfield (ISBN 0-521-58519-8, Cambrigde University Press, 1997). O segundo livro também inspirou outros exercícios sobre strings.