6.14 Conclusão

Neste capítulo, você aprendeu mais sobre declarações de método. Você também aprendeu a diferença entre métodos de instâncias e os métodos `static` e como chamar métodos `static` precedendo o nome do método com o nome da classe em que ele aparece e um ponto (`.`) separador. Aprendeu a utilizar os operadores `+` e `+=` para realizar concatenações de string. Discutimos como a pilha de chamadas de método e os registros de ativação monitoram os métodos que foram chamados e para onde cada método deve retornar quando ele completa a sua tarefa. Também discutimos as regras de promoção do Java para converter implicitamente entre tipos primitivos e como realizar conversões explícitas com operadores de coerção. Em seguida, você aprendeu sobre alguns dos pacotes mais utilizados na Java API.

Você aprendeu a declarar constantes identificadas utilizando tanto tipos `enum` como variáveis `private static final`. Você utilizou a classe `SecureRandom` para gerar números aleatórios para simulações. Você também aprendeu o escopo dos campos e variáveis locais em uma classe. Por fim, você aprendeu que múltiplos métodos em uma classe podem ser sobrecarregados fornecendo ao método o mesmo nome e assinaturas diferentes. Esses métodos podem ser utilizados para realizar as mesmas tarefas, ou tarefas semelhantes, utilizando tipos diferentes ou números distintos de parâmetros.

No Capítulo 7, você aprenderá a manter listas e tabelas de dados em arrays. Veremos uma implementação mais elegante do aplicativo que rola um dado 6.000.000 vezes. Apresentaremos duas versões de um estudo de caso `GradeBook` que armazena conjuntos das notas de alunos em um objeto `GradeBook`. Você também aprenderá a acessar os argumentos de linha de comando de um aplicativo que são passados para o método `main` quando um aplicativo começa a execução.

Resumo

Seção 6.1 Introdução

- A experiência mostrou que a melhor maneira de desenvolver e manter um programa grande é construí-lo a partir de pequenos e simples pedaços, ou módulos. Essa técnica é chamada dividir para conquistar.

Seção 6.2 Módulos de programa em Java

- Os métodos são declarados dentro de classes. Em geral, as classes são agrupadas em pacotes para que possam ser importadas e reutilizadas.
- Os métodos permitem modularizar um programa separando suas tarefas em unidades autocontidas. As instruções em um método são escritas somente uma vez e permanecem ocultas de outros métodos.
- Usar os métodos existentes como blocos de construção para criar novos programas é uma forma de reutilização de software que permite evitar repetição de código dentro de um programa.

Seção 6.3 Métodos `static`, campos `static` e classe `Math`

- Uma chamada de método especifica o nome do método a ser chamado e fornece os argumentos que o método chamado requer para realizar sua tarefa. Quando a chamada de método é concluída, o método retorna um resultado, ou simplesmente o controle, ao seu chamador.
- Uma classe pode conter métodos `static` para realizar tarefas comuns que não exigem um objeto da classe. Quaisquer dados que um método `static` poderia requerer para realizar suas tarefas podem ser enviados ao método como argumentos em uma chamada de método. Um método `static` é chamado especificando o nome da classe em que o método é declarado seguido por um ponto (`.`) e pelo nome do método, como em

  NomeDaClasse`.`*nomeDoMétodo*`(`*argumentos*`)`

- A classe `Math` fornece os métodos `static` para realizar cálculos matemáticos comuns.
- A constante `Math.PI` (3,141592653589793) é a relação entre a circunferência de um círculo e seu diâmetro. A constante `Math.E` (2,718281828459045) é o valor base para logaritmos naturais (calculados com o método `static` `Math` `log`).
- `Math.PI` e `Math.E` são declaradas com os modificadores `public`, `final` e `static`. Torná-los `public` permite que você use esses campos nas suas próprias classes. Um campo declarado com a palavra-chave `final` é constante — seu valor não pode ser alterado depois de ele ser inicializado. Tanto `PI` como `E` são declarados `final` porque seus valores nunca mudam. Tornar esses campos `static` permite que eles sejam acessados pelo nome da classe `Math` e um ponto (`.`) separador, como ocorre com os métodos da classe `Math`.
- Todos os objetos de uma classe compartilham uma cópia dos campos `static` da classe. As variáveis de classe e as variáveis de instância representam os campos de uma classe.
- Quando você executa a Java Virtual Machine (JVM) com o comando `java`, a JVM carrega a classe especificada e utiliza esse nome de classe para invocar o método `main`. É possível especificar argumentos de linha de comando adicionais que a JVM passará para o seu aplicativo.
- Você pode colocar um método `main` em cada classe que declara — somente o método `main` na classe que você utiliza para executar o aplicativo será chamado pelo comando `java`.

Seção 6.4 Declarando métodos com múltiplos parâmetros

- Quando um método é chamado, o programa faz uma cópia dos valores de argumento do método e os atribui aos parâmetros correspondentes do método. Quando o controle do programa retorna ao ponto em que método foi chamado, os parâmetros do método são removidos da memória.
- Um método pode retornar no máximo um valor, mas o valor retornado poderia ser uma referência a um objeto que contém muitos valores.
- Variáveis devem ser declaradas como campos de uma classe somente se forem utilizadas em mais de um método da classe ou se o programa deve salvar seus valores entre chamadas aos métodos da classe.
- Se um método tiver mais de um parâmetro, os parâmetros serão especificados como uma lista separada por vírgulas. Deve haver um argumento na chamada de método para cada parâmetro na declaração do método. Além disso, cada argumento deve ser consistente com o tipo do parâmetro correspondente. Se um método não aceitar argumentos, a lista de parâmetros ficará vazia.
- `Strings` podem ser concatenadas com o operador +, o que posiciona os caracteres do operando direito no final daqueles no operando esquerdo.
- Cada objeto e valor primitivos no Java podem ser representados como uma `String`. Quando um objeto é concatenado com uma `String`, ele é convertido em uma `String` e então as duas `Strings` são concatenadas.
- Se um `boolean` for concatenado com uma `String`, a palavra "`true`" ou "`false`" é utilizada para representar o valor `boolean`.
- Todos os objetos em Java têm um método especial chamado `toString` que retorna uma representação `String` do conteúdo do objeto. Quando um objeto é concatenado com uma `String`, a JVM chama implicitamente o método `toString` do objeto a fim de obter a representação string do objeto.
- Pode-se dividir grandes literais `String` em várias `Strings` menores e colocá-las em múltiplas linhas de código para melhorar a legibilidade, depois remontar as `Strings` utilizando concatenação.

Seção 6.5 Notas sobre a declaração e utilização de métodos

- Há três maneiras de chamar um método — utilizar o próprio nome de um método para chamar um outro método da mesma classe; utilizar uma variável que contém uma referência a um objeto, seguido por um ponto (`.`) e o nome do método para chamar um método do objeto referenciado; e utilizar o nome da classe e um ponto (`.`) para chamar um método `static` de uma classe.
- Há três maneiras de retornar o controle a uma instrução que chama um método. Se o método não retornar um resultado, o controle retornará quando o fluxo do programa alcançar a chave direita de fechamento do método ou quando a instrução

  ```
  return;
  ```

  for executada. Se o método retornar um resultado, a instrução

  ```
  return expressão;
  ```

  avalia a *expressão* e então imediatamente retorna o valor resultante ao chamador.

Seção 6.6 Pilhas de chamadas de método e quadros de pilha

- As pilhas são conhecidas como estruturas de dados do tipo último a entrar, primeiro a sair (*last-in, first-out* —— LIFO) — o último item inserido na pilha é o primeiro item que é removido dela.
- Um método chamado deve saber como retornar ao seu chamador, portanto o endereço de retorno do método de chamada é colocado na pilha de chamadas de método quando o método for chamado. Se uma série de chamadas de método ocorrer, os sucessivos endereços de retorno são empilhados na ordem "último a entrar, primeiro a sair", de modo que o último método a executar será o primeiro a retornar ao seu chamador.
- A pilha de chamadas de método contém a memória para as variáveis locais utilizadas em cada invocação de um método durante a execução de um programa. Esses dados são conhecidos como registro de ativação ou quadro de pilha da chamada de método. Quando uma chamada de método é feita, o quadro de pilha para ela é colocado na pilha de chamadas de método. Quando o método retorna ao seu chamador, a sua chamada do registro de ativação é retirada da pilha e as variáveis locais não são mais conhecidas para o programa.
- Se mais chamadas de método forem feitas do que o quadro de pilha pode armazenar na pilha de chamadas de método, ocorre um erro conhecido como estouro de pilha. O aplicativo compilará corretamente, mas sua execução causa um estouro de pilha.

Seção 6.7 Promoção e coerção de argumentos

- A promoção de argumentos converte o valor de um argumento para o tipo que o método espera receber no parâmetro correspondente.
- Regras de promoção se aplicam a expressões que contêm valores de dois ou mais tipos primitivos e a valores de tipo primitivo passados como argumentos para os métodos. Cada valor é promovido para o tipo "mais alto" na expressão. Em casos em que as informações podem ser perdidas por causa da conversão, o compilador Java exige que se utilize um operador de coerção para forçar explicitamente que a conversão ocorra.

Seção 6.9 Estudo de caso: geração segura de números aleatórios

- Objetos da classe `SecureRandom` (pacote `java.security`) podem produzir valores aleatórios não determinísticos.
- O método `nextInt` `SecureRandom` gera um valor aleatório.

- A classe `SecureRandom` fornece uma outra versão do método `nextInt` que recebe um argumento `int` e retorna um valor a partir de 0, mas sem incluí-lo, até o valor do argumento.
- Números aleatórios em um intervalo podem ser gerados com

```
int number = valorDeDeslocamento + randomNumbers.nextInt(fatorDeEscalonamento);
```

  onde *valorDeDeslocamento* especifica o primeiro número no intervalo desejado de inteiros consecutivos e *fatorDeEscalonamento* especifica quantos números estão no intervalo.
- Os números aleatórios podem ser escolhidos a partir de intervalos de inteiro não consecutivos, como em

```
int number = valorDeDeslocamento +
   diferençaEntreValores * randomNumbers.nextInt(fatorDeEscalonamento);
```

  onde *valorDeDeslocamento* especifica o primeiro número no intervalo de valores, *diferençaEntreValores* representa a diferença entre números consecutivos na sequência e *fatorDeEscalonamento* especifica quantos números estão no intervalo.

Seção 6.10 Estudo de caso: um jogo de azar; apresentando tipos `enum`

- Um tipo `enum` é introduzido pela palavra-chave `enum` e um nome de tipo. Como com qualquer classe, as chaves (`{` e `}`) delimitam o corpo de uma declaração `enum`. Entre as chaves há uma lista de constantes `enum`, cada uma representando um valor único separado por vírgula. Os identificadores em uma `enum` devem ser únicos. Pode-se atribuir variáveis de um tipo `enum` somente a constantes do tipo `enum`.
- Constantes também podem ser declaradas como variáveis `private static final`. Essas constantes, por convenção, são declaradas com todas as letras maiúsculas fazendo com que elas se destaquem no programa.

Seção 6.11 Escopo das declarações

- O escopo é a parte do programa em que uma entidade, como uma variável ou um método, pode ser referida pelo seu nome. Diz-se que essa entidade está "no escopo" para essa parte do programa.
- O escopo de uma declaração de parâmetro é o corpo do método em que a declaração aparece.
- O escopo de uma declaração de variável local é do ponto em que a declaração aparece até o final desse bloco.
- O escopo de uma declaração de variável local que aparece na seção de inicialização do cabeçalho de uma instrução `for` é o corpo da instrução `for` e as outras expressões no cabeçalho.
- O escopo de um método ou campo de uma classe é o corpo inteiro da classe. Isso permite que os métodos da classe utilizem nomes simples para chamar os outros métodos da classe e acessem os campos da classe.
- Qualquer bloco pode conter declarações de variável. Se uma variável local ou um parâmetro em um método tiver o mesmo nome de um campo, este permanece sombreado até que o bloco termine a execução.

Seção 6.12 Sobrecarga de método

- O Java permite métodos sobrecarregados em uma classe, desde que os métodos tenham diferentes conjuntos de parâmetros (determinados pelo número, ordem e tipo de parâmetros).
- Métodos sobrecarregados são distinguidos por suas assinaturas — combinações dos nomes e número, tipos e ordem dos parâmetros dos métodos, mas não pelos tipos de retorno.

Exercícios de revisão

6.1 Preencha as lacunas em cada uma das seguintes afirmações:

a) Um método é invocado com um(a) _____.

b) Uma variável conhecida somente dentro do método em que é declarada chama-se _____.

c) A instrução _____ em um método chamado pode ser utilizada para passar o valor de uma expressão de volta para o método de chamada.

d) A palavra-chave _____ indica que um método não retorna um valor.

e) Os dados podem ser adicionados ou removidos somente do(a) _____ de uma pilha.

f) As pilhas são conhecidas como estruturas de dados _____; o último item colocado (inserido) na pilha é o primeiro item retirado (removido) da pilha.

g) As três maneiras de retornar o controle de um método chamado a um chamador são _____, _____ e _____.

h) Um objeto da classe _____ produz números verdadeiramente aleatórios.

i) A pilha de execução de programas contém a memória criada para variáveis locais a cada invocação de método durante a execução de um programa. Esses dados, armazenados como parte da pilha de chamadas de método, são conhecidos como _____ ou _____ da chamada de método.

j) Se houver mais chamadas de método do que pode ser armazenado na pilha de execução do programa, um erro conhecido como _____ ocorrerá.

k) O _____ de uma declaração é a parte de um programa que pode referenciar a entidade na declaração pelo nome.

l) É possível ter diversos métodos com o mesmo nome que operam, separadamente, sobre diferentes tipos ou números de argumentos. Esse recurso é chamado de _____.

6.2 Para a classe `Craps` na Figura 6.8, declare o escopo de cada uma das seguintes entidades:

a) a variável `randomNumbers`.

b) a variável `die1`.

c) o método `rollDice`.

d) o método `main`.

e) a variável `sumOfDice`.

6.3 Escreva um aplicativo que teste se os exemplos de chamadas de método da classe `Math` mostrada na Figura 6.2 realmente produzem os resultados indicados.

6.4 Forneça o cabeçalho de método para cada um dos seguintes métodos.

a) O método `hypotenuse`, que aceita dois argumentos de ponto flutuante de precisão dupla `side1` e `side2` e retorna um resultado de ponto flutuante de dupla precisão.

b) O método `smallest`, que recebe três inteiros `x`, `y` e `z` e retorna um inteiro.

c) O método `instructions`, que não aceita nenhum argumento e não retorna um valor. [*Observação:* esses métodos são comumente utilizados para exibição de instruções para o usuário.]

d) O método `intToFloat`, que recebe um argumento `number` do tipo inteiro e retorna um `float`.

6.5 Encontre o erro em cada um dos seguintes segmentos de programa. Explique como corrigir o erro.

a)
```
void g()
{
       System.out.println("Inside method g");

       void h()
       {
          System.out.println("Inside method h");
       }
}
```

b)
```
int sum(int x, int y)
{
       int result;
       result = x + y;
}
```

c)
```
void f(float a);
{
       float a;
       System.out.println(a);
}
```

d)
```
void product()
{
       int a = 6, b = 5, c = 4, result;
       result = a * b * c;
       System.out.printf("Result is %d%n", result);
       return result;
}
```

6.6 Declare o método `sphereVolume` para calcular e retornar o volume da esfera. Utilize a seguinte instrução para calcular o volume:

```
double volume = (4.0 / 3.0) * Math.PI * Math.pow(radius, 3)
```

Escreva um aplicativo Java que solicita ao usuário o raio do tipo `double` de uma esfera, chama `sphereVolume` para calcular o volume e exibe o resultado.

Respostas dos exercícios de revisão

6.1 a) chamada de método. b) variável local. c) `return`. d) `void`. e) parte superior. f) último a entrar, primeiro a sair (LIFO). g) `return;` ou `return` *expressão*`;` ou encontre a chave direita de fechamento de um método. h) `SecureRandom`. i) registro de ativação, quadro de pilha. j) estouro de pilha. k) escopo. l) sobrecarga de método.

6.2 a) corpo de classe. b) bloco que define o corpo do método `rollDice`. c) corpo de classe. d) corpo de classe. e) bloco que define o corpo do método `main`.

6.3 A seguinte solução demonstra os métodos da classe `Math` na Figura 6.2:

```
1  // Exercício 6.3: MathTest.java
2  // Testando os métodos da classe Math
3  public class MathTest
4  {
5    public static void main(String[] args)
6    {
7          System.out.printf("Math.abs(23.7) = %f%n", Math.abs(23.7));
8      System.out.printf("Math.abs(0.0) = %f%n", Math.abs(0.0));
9      System.out.printf("Math.abs(-23.7) = %f%n", Math.abs(-23.7));
10     System.out.printf("Math.ceil(9.2) = %f%n", Math.ceil(9.2));
11     System.out.printf("Math.ceil(-9.8) = %f%n", Math.ceil(-9.8));
12     System.out.printf("Math.cos(0.0) = %f%n", Math.cos(0.0));
13     System.out.printf("Math.exp(1.0) = %f%n", Math.exp(1.0));
14     System.out.printf("Math.exp(2.0) = %f%n", Math.exp(2.0));
15     System.out.printf("Math.floor(9.2) = %f%n", Math.floor(9.2));
16     System.out.printf("Math.floor(-9.8) = %f%n", Math.floor(-9.8));
17     System.out.printf("Math.log(Math.E) = %f%n", Math.log(Math.E));
18     System.out.printf("Math.log(Math.E * Math.E) = %f%n",
19       Math.log(Math.E * Math.E));
20     System.out.printf("Math.max(2.3, 12.7) = %f%n", Math.max(2.3, 12.7));
21     System.out.printf("Math.max(-2.3, -12.7) = %f%n",
22       Math.max(-2.3, -12.7));
23     System.out.printf("Math.min(2.3, 12.7) = %f%n", Math.min(2.3, 12.7));
24     System.out.printf("Math.min(-2.3, -12.7) = %f%n",
25       Math.min(-2.3, -12.7));
26     System.out.printf("Math.pow(2.0, 7.0) = %f%n", Math.pow(2.0, 7.0));
27     System.out.printf("Math.pow(9.0, 0.5) = %f%n", Math.pow(9.0, 0.5));
28     System.out.printf("Math.sin(0.0) = %f%n", Math.sin(0.0));
29     System.out.printf("Math.sqrt(900.0) = %f%n", Math.sqrt(900.0));
30     System.out.printf("Math.tan(0.0) = %f%n", Math.tan(0.0));
31   } // fim de main
32 } // fim da classe MathTest
```

```
Math.abs(23.7) = 23.700000
Math.abs(0.0) = 0.000000
Math.abs(-23.7) = 23.700000
Math.ceil(9.2) = 10.000000
Math.ceil(-9.8) = -9.000000
Math.cos(0.0) = 1.000000
Math.exp(1.0) = 2.718282
Math.exp(2.0) = 7.389056
Math.floor(9.2) = 9.000000
Math.floor(-9.8) = -10.000000
Math.log(Math.E) = 1.000000
Math.log(Math.E * Math.E) = 2.000000
Math.max(2.3, 12.7) = 12.700000
Math.max(-2.3, -12.7) = -2.300000
Math.min(2.3, 12.7) = 2.300000
Math.min(-2.3, -12.7) = -12.700000
Math.pow(2.0, 7.0) = 128.000000
Math.pow(9.0, 0.5) = 3.000000
Math.sin(0.0) = 0.000000
Math.sqrt(900.0) = 30.000000
Math.tan(0.0) = 0.000000
```

6.4 a) `double hypotenuse(double side1, double side2)`

b) `int smallest(int x, int y, int z)`

c) `void instructions()`

d) `float intToFloat(int number)`

6.5 a) Erro: o método `h` é declarado dentro do método `g`.
Correção: mova a declaração de `h` para fora da declaração de `g`.

b) Erro: o método supostamente deve retornar um inteiro, mas não o faz.
Correção: exclua a variável `result` e coloque a instrução

```
return x + y;
```

no método ou adicione a seguinte instrução no fim do corpo de método:

```
return result;
```

c) Erro: ponto e vírgula após o parêntese direito da lista de parâmetros está incorreto e o parâmetro a não deve ser redeclarado no método.
Correção: exclua o ponto e vírgula após o parêntese direito da lista de parâmetros e exclua a declaração `float a;`.

d) Erro: o método retorna um valor quando supostamente não deveria.
Correção: altere o tipo de retorno de `void` para `int`.

6.6 A solução a seguir calcula o volume de uma esfera, utilizando o raio inserido pelo usuário:

```
1  // Exercício 6.6: Sphere.java
2  // Calcula o volume de uma esfera.
3  import java.util.Scanner;
4
5  public class Sphere
6  {
7    // obtém o raio a partir do usuário e exibe o volume da esfera
8    public static void main(String[] args)
9    {
10     Scanner input = new Scanner(System.in);
11
12     System.out.print("Enter radius of sphere: ");
13     double radius = input.nextDouble();
14
15     System.out.printf("Volume is %f%n", sphereVolume(radius));
16   } // fim do método determineSphereVolume
17
18   // calcula e retorna volume de esfera
19   public static double sphereVolume(double radius)
20   {
21     double volume = (4.0 / 3.0) * Math.PI * Math.pow(radius, 3);
22     return volume;
23   } // fim do método sphereVolume
24 } // fim da classe Sphere
```

```
Enter radius of sphere: 4
Volume is 268.082573
```

Questões

6.7 Qual é o valor de x depois que cada uma das seguintes instruções é executada?

a) `x = Math.abs(7.5);`

b) `x = Math.floor(7.5);`

c) `x = Math.abs(0.0);`

d) `x = Math.ceil(0.0);`

e) `x = Math.abs(-6.4);`

f) `x = Math.ceil(-6.4);`

g) `x = Math.ceil(-Math.abs(-8 + Math.floor(-5.5)));`

6.8 (***Taxas de estacionamento***) Um estacionamento cobra uma tarifa mínima de R$ 2,00 para estacionar por até três horas. Um adicional de R$ 0,50 por hora *não necessariamente inteira* é cobrado após as três primeiras horas. A tarifa máxima para qualquer dado período de 24 horas é R$ 10,00. Suponha que nenhum carro fique estacionado por mais de 24 horas por vez. Escreva um aplicativo que calcule e exiba as tarifas de estacionamento para cada cliente que estacionou nessa garagem ontem. Você deve inserir as horas de estacionamento para cada cliente. O programa deve exibir a cobrança para o cliente atual e calcular e exibir o total dos recibos de ontem. Ele deve utilizar o método `calculateCharges` para determinar a tarifa para cada cliente.

6.9 (***Arredondando números***) `Math.floor` pode ser utilizado para arredondar valores ao número inteiro mais próximo — por exemplo,

```
y = Math.floor(x + 0.5);
```

arredondará o número x para o inteiro mais próximo e atribuirá o resultado a y. Escreva um aplicativo que lê valores `double` e utiliza a instrução anterior para arredondar cada um dos números para o inteiro mais próximo. Para cada número processado, exiba ambos os números, o original e o arredondado.

6.10 (***Arredondando números***) Para arredondar números em casas decimais específicas, utilize uma instrução como

```
y = Math.floor(x * 10 + 0.5) / 10;
```

que arredonda x para a casa decimal (isto é, a primeira posição à direita do ponto de fração decimal), ou

```
y = Math.floor(x * 100 + 0.5) / 100;
```

que arredonda x para a casa centesimal (isto é, a segunda posição à direita do ponto de fração decimal). Escreva um aplicativo que defina quatro métodos para arredondar um número x de várias maneiras:

a) `roundToInteger(number)`

b) `roundToTenths(number)`

c) `roundToHundredths(number)`

d) `roundToThousandths(number)`

Para cada leitura de valor, seu programa deve exibir o valor original, o número arredondado para o inteiro mais próximo, o número arredondado para o décimo mais próximo, o número arredondado para o centésimo mais próximo e o número arredondado para o milésimo mais próximo.

6.11 Responda cada uma das seguintes perguntas:

a) O que significa escolher números "aleatoriamente"?

b) Por que o método `nextInt` da classe `SecureRandom` é útil para simular jogos de azar?

c) Por que frequentemente é necessário escalonar ou deslocar os valores produzidos por um objeto `SecureRandom`?

d) Por que a simulação computadorizada de situações do mundo real é uma técnica útil?

6.12 Escreva instruções que atribuem inteiros aleatórios à variável n nos seguintes intervalos:

a) $1 \leq n \leq 2$.

b) $1 \leq n \leq 100$.

c) $0 \leq n \leq 9$.

d) $1000 \leq n \leq 1112$.

e) $-1 \leq n \leq 1$.

f) $-3 \leq n \leq 11$.

6.13 Escreva instruções que exibirão um número aleatório de cada um dos seguintes conjuntos:

a) 2, 4, 6, 8, 10.

b) 3, 5, 7, 9, 11.

c) 6, 10, 14, 18, 22.

6.14 (***Exponenciação***) Escreva um método `integerPower(base, exponent)` que retorne o valor de

$base^{exponente}$

Por exemplo, `integerPower(3, 4)` calcula 3^4 (ou `3 * 3 * 3 * 3`). Suponha que `exponent` seja um inteiro não zero, positivo, e `base`, um inteiro. Use uma instrução `for` ou `while` para controlar o cálculo. Não utilize métodos da classe `Math`. Incorpore esse método a um aplicativo que lê os valores inteiros para `base` e `exponent` e realiza o cálculo com o método `integerPower`.

6.15 (***Cálculos de hipotenusa***) Defina um método `hypotenuse` que calcula a hipotenusa de um triângulo retângulo quando são dados os comprimentos dos outros dois lados. O método deve tomar dois argumentos do tipo `double` e retornar a hipotenusa como um `double`. Incorpore esse método a um aplicativo que lê valores para `side1` e `side2` e realiza o cálculo com o método `hypotenuse`. Utilize os métodos `Math` `pow` e `sqrt` para determinar o tamanho da hipotenusa de cada um dos triângulos na Figura 6.15. [*Observação:* a classe `Math` também fornece o método `hypot` para realizar esse cálculo.]

Triângulo	Lado 1	Lado 2
1	3.0	4.0
2	5.0	12.0
3	8.0	15.0

Figura 6.15 | Valores para os lados dos triângulos na Questão 6.15.

6.16 (***Múltiplos***) Escreva um método `isMultiple` que determina um par de inteiros se o segundo inteiro for um múltiplo do primeiro. O método deve aceitar dois argumentos inteiros e retornar `true` se o segundo for um múltiplo do primeiro e `false` caso contrário. [*Dica:* utilize o operador de módulo.] Incorpore esse método a um aplicativo que insere uma série de pares inteiros (um par por vez) e determina se o segundo valor em cada par é um múltiplo do primeiro.

6.17 (***Par ou ímpar***) Escreva um método `isEven` que utiliza o operador de resto (%) para determinar se um inteiro é par. O método deve levar um argumento inteiro e retornar `true` se o número inteiro for par, e `false`, caso contrário. Incorpore esse método a um aplicativo que insere uma sequência de inteiros (um por vez) e determina se cada um é par ou ímpar.

6.18 (***Exibindo um quadrado de asteriscos***) Escreva um método `squareOfAsterisks` que exibe um quadrado sólido (o mesmo número de linhas e colunas) de asteriscos cujo lado é especificado no parâmetro inteiro `side`. Por exemplo, se `side` for 4, o método deverá exibir

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

Incorpore esse método a um aplicativo que lê um valor inteiro para `side` a partir da entrada fornecida pelo usuário e gera saída dos asteriscos com o método `squareOfAsterisks`.

6.19 (***Exibindo um quadrado de qualquer caractere***) Modifique o método criado no Exercício 6.18 para receber um segundo parâmetro do tipo `char` chamado `fillCharacter`. Forme o quadrado utilizando o `char` fornecido como um argumento. Portanto, se `side` for `5` e `fillCharacter` for `#`, o método deve exibir

```
#####
#####
#####
#####
#####
```

Utilize a seguinte instrução (em que `input` é um objeto `Scanner`) para ler um caractere do usuário no teclado:

```
char fill = input.next().charAt(0);
```

6.20 (***Área de círculo***) Escreva um aplicativo que solicita ao usuário o raio de um círculo e utiliza um método chamado `circleArea` para calcular a área do círculo.

6.21 (***Separando dígitos***) Escreva métodos que realizam cada uma das seguintes tarefas:

a) Calcule a parte inteiro do quociente quando o inteiro `a` é dividido pelo inteiro `b`.

b) Calcule o resto inteiro quando o inteiro `a` é dividido por inteiro `b`.

c) Utilize métodos desenvolvidos nas partes (a) e (b) para escrever um método `displayDigits` que recebe um inteiro entre `1` e `99999` e o exibe como uma sequência de dígitos, separando cada par de dígitos por dois espaços. Por exemplo, o inteiro `4562` deve aparecer como

```
4  5  6  2
```

Incorpore os métodos em um aplicativo que insere um número inteiro e chama `displayDigits` passando para o método o número inteiro inserido. Exiba os resultados.

6.22 (***Conversões de temperatura***) Implemente os seguintes métodos inteiros:

a) O método `celsius` retorna o equivalente em Celsius de uma temperatura em Fahrenheit utilizando o cálculo

```
celsius = 5.0 / 9.0 * (fahrenheit - 32);
```

b) O método `fahrenheit` retorna o equivalente em Fahrenheit de uma temperatura em Celsius utilizando o cálculo

```
fahrenheit = 9.0 / 5.0 * celsius + 32;
```

c) Utilize os métodos nas partes (a) e (b) para escrever um aplicativo que permite ao usuário inserir uma temperatura em Fahrenheit e exibir o equivalente em Celsius ou inserir uma temperatura em Celsius e exibir o equivalente em Fahrenheit.

6.23 (***Localize o mínimo***) Escreva um método `minimum3` que retorna o menor dos três números de ponto flutuante. Utilize o método `Math.min` para implementar `minimum3`. Incorpore o método a um aplicativo que lê três valores do usuário, determina o menor valor e exibe o resultado.

6.24 (***Números perfeitos***) Dizemos que um número inteiro é um *número perfeito* se a soma de seus fatores, incluindo 1 (mas não o próprio número), for igual ao número. Por exemplo, 6 é um número perfeito porque 6 = 1 + 2 + 3. Escreva um método `isPerfect` que determina se parâmetro `number` é um número perfeito. Utilize esse método em um applet que determina e exibe todos os números perfeitos entre 1 e 1.000. Exiba os fatores de cada número perfeito confirmando que ele é de fato perfeito. Desafie o poder de computação do seu computador testando números bem maiores que 1.000. Exiba os resultados.

6.25 (***Números primos***) Um número inteiro positivo é *primo* se for divisível apenas por 1 e por ele mesmo. Por exemplo, 2, 3, 5 e 7 são primos, mas 4, 6, 8 e 9 não são. O número 1, por definição, não é primo.

a) Escreva um método que determina se um número é primo.

b) Utilize esse método em um aplicativo que determina e exibe todos os números primos menores que 10.000. Quantos números até 10.000 você precisa testar a fim de assegurar que encontrou todos os primos?

c) Inicialmente, você poderia pensar que $n/2$ é o limite superior que deve ser testado para ver se um número é primo, mas você precisa ir apenas até a raiz quadrada de n. Reescreva o programa e execute-o de ambas as maneiras.

6.26 **(*Invertendo dígitos*)** Escreva um método que recebe um valor inteiro e retorna o número com seus dígitos invertidos. Por exemplo, dado o número 7.631, o método deve retornar 1.367. Incorpore o método a um aplicativo que lê um valor a partir da entrada fornecida pelo usuário e exibe o resultado.

6.27 **(*Máximo divisor comum*)** O *máximo divisor comum (MDC)* de dois inteiros é o maior inteiro que é divisível por cada um dos dois números. Escreva um método `mdc` que retorna o máximo divisor comum de dois inteiros. [*Dica:* você poderia querer utilizar o algoritmo de Euclides. Você pode encontrar informações sobre isso em `en.wikipedia.org/wiki/Euclidean_algorithm`.] Incorpore o método a um aplicativo que lê dois valores do usuário e exibe o resultado.

6.28 Escreva um método `qualityPoints` que insere a média de um aluno e retorna `4` se for 90 a 100, `3` se 80 a 89, `2` se 70 a 79, `1` se 60 a 69 e `0` se menor que 60. Incorpore o método a um aplicativo que lê um valor a partir do usuário e exibe o resultado.

6.29 **(*Cara ou coroa*)** Escreva um aplicativo que simula o jogo de cara ou coroa. Deixe o programa lançar uma moeda toda vez que o usuário escolher a opção "`Toss Coin`" no menu. Conte o número de vezes que cada lado da moeda aparece. Exiba os resultados. O programa deve chamar um método `flip` separado que não aceita argumentos e retorna um valor a partir de um `Coin enum` (`HEADS` e `TAILS`). [*Observação*: se o programa simular de modo realista o arremesso de moeda, cada lado da moeda deve aparecer aproximadamente metade das vezes.]

6.30 **(*Adivinhe o número*)** Escreva um aplicativo que execute "adivinhe o número" como mostrado a seguir: seu programa escolhe o número a ser adivinhado selecionando um inteiro aleatório no intervalo de 1 a 1.000. O aplicativo exibe o prompt `Guess a number between 1 and 1000 [adivinhe um número entre 1 e 1000]`. O jogador insere uma primeira suposição. Se o palpite do jogador estiver incorreto, seu programa deve exibir `Too high. Try again [Muito alto. Tente novamente]` ou `Too low. Try again [Muito baixo. Tente novamente]` para ajudar o jogador a alcançar a resposta correta. O programa deve solicitar ao usuário o próximo palpite. Quando o usuário insere a resposta correta, exibe `Congratulations. You guessed the number. [Parabéns, você adivinhou o número!]` e permite que o usuário escolha se quer jogar novamente. [*Observação*: a técnica de adivinhação empregada nesse problema é semelhante a uma pesquisa binária, discutida no Capítulo 19, "Pesquisa, classificação e Big O".]

6.31 **(*Adivinhe a modificação de número*)** Modifique o programa do Exercício 6.30 para contar o número de adivinhações que o jogador faz. Se o número for 10 ou menos, exibe `Either you know the secret or you got lucky! [Você sabe o segredo ou tem muita sorte!]`; se o jogador adivinhar o número em 10 tentativas, exiba `Aha! You know the secret! [Aha! Você sabe o segredo!]`; se o jogador fizer mais que 10 adivinhações, exiba `You should be able to do better! [Você deve ser capaz de fazer melhor]`. Por que esse jogo não deve precisar de mais que 10 suposições? Bem, com cada "boa adivinhação" o jogador deve ser capaz de eliminar a metade dos números, depois a metade dos números restantes, e assim por diante.

6.32 **(*Distância entre pontos*)** Escreva um método `distance` para calcular a distância entre dois pontos ($x1$, $y1$) e ($x2$, $y2$). Todos os números e valores de retorno devem ser do tipo `double`. Incorpore esse método a um aplicativo que permite que o usuário insira as coordenadas de pontos.

6.33 **(*Modificação do jogo Craps*)** Modifique o programa de jogo de dados craps da Figura 6.8 para permitir apostas. Inicialize a variável `bankBalance` como 1.000 dólares. Peça ao jogador que insira um `wager`. Verifique se `wager` é menor ou igual a `bankBalance` e, se não for, faça o usuário reinserir `wager` até um `wager` válido ser inserido. Então, execute um jogo de dados. Se o jogador ganhar, aumente `bankBalance` por `wager` e exiba o novo `bankBalance`. Se o jogador perder, diminua `bankBalance` por `wager`, exiba o novo `bankBalance`, verifique se `bankBalance` tornou-se zero e, se isso tiver ocorrido, exiba a mensagem `"Sorry. You busted!" ["Desculpe, mas você faliu!"]`. À medida que o jogo se desenvolve, exiba várias mensagens para criar uma "conversa", como `"Oh, you're going for broke, huh?" ["Oh, parece que você vai quebrar, hein?"]` ou `"Aw c'mon, take a chance!" ["Ah, vamos lá, dê uma chance para sua sorte"]` ou `"You're up big. Now's the time to cash in your chips!" [Você está montado na grana. Agora é hora de trocar essas fichas e embolsar o dinheiro!"]`. Implemente a "conversa" como um método separado que escolhe aleatoriamente a string a ser exibida.

6.34 **(*Tabela de números binários, octais e hexadecimais*)** Escreva um aplicativo que exibe uma tabela de equivalentes binários, octais e hexadecimais dos números decimais no intervalo de 1 a 256. Se você não estiver familiarizado com esses sistemas de números, leia primeiro o Apêndice J, em inglês, na Sala Virtual do Livro.

Fazendo a diferença

À medida que o preço dos computadores cai, torna-se viável para cada estudante, apesar da circunstância econômica, ter um computador e utilizá-lo na escola. Isso cria grandes oportunidades para aprimorar a experiência educativa de todos os estudantes em todo o mundo, conforme sugerido pelos cinco exercícios a seguir. [*Observação*: verifique iniciativas como One Laptop Per Child Project (`www.laptop.org`). Pesquise também laptops "verdes" — quais são as principais características amigáveis ao meio ambiente desses dispositivos? Consulte a Electronic Product Environmental Assessment Tool (`www.epeat.net`), que pode ajudar a avaliar o grau de responsabilidade ambiental "greenness" de computadores desktop, notebooks e monitores para ajudar a decidir que produtos comprar.]

6.35 **(*Instrução assistida por computador*)** O uso de computadores na educação é chamado *instrução assistida por computador* (CAI). Escreva um programa que ajudará um aluno da escola elementar a aprender multiplicação. Utilize um objeto `SecureRandom` para produzir dois inteiros positivos de um algarismo. O programa deve então fazer ao usuário uma pergunta, como

```
Quanto é 6 vezes 7?
```

O aluno insere então a resposta. Em seguida, o programa verifica a resposta do aluno. Se estiver correta, exiba a mensagem "`Muito bem!`" e faça uma outra pergunta de multiplicação. Se a resposta estiver errada, exiba a mensagem "`Não. Por favor, tente de novo.`" e deixe que o aluno tente a mesma pergunta várias vezes até que por fim ele acerte. Um método separado deve ser utilizado para gerar cada nova pergunta. Esse método deve ser chamado uma vez quando a aplicação inicia a execução e toda vez que o usuário responde a pergunta corretamente.

6.36 **(*Instrução auxiliada por computador: reduzindo a fadiga do aluno*)** Um problema em ambientes CAI é a fadiga do aluno. Isso pode ser reduzido variando-se as respostas do computador para prender a atenção do aluno. Modifique o programa da Questão 6.35 para que vários comentários sejam exibidos para cada resposta como mostrado a seguir:

Possibilidades para uma resposta correta:

```
Muito bom!
Excelente!
Bom trabalho!
Mantenha um bom trabalho!
```

Possibilidades para uma resposta incorreta:

```
Não. Por favor, tente de novo.
Errado. Tente mais uma vez.
Não desista!
Não. Continue tentando.
```

Utilize a geração de números aleatórios para escolher um número de 1 a 4 que será utilizado para selecionar uma de quatro respostas adequadas a cada resposta correta ou incorreta. Utilize uma instrução `switch` para emitir as respostas.

6.37 **(*Instrução auxiliada por computador: monitorando o desempenho do aluno*)** Sistemas mais sofisticados de instruções auxiliadas por computador monitoram o desempenho do aluno durante um período de tempo. A decisão sobre um novo tópico frequentemente é baseada no sucesso do aluno com tópicos prévios. Modifique o programa de Exercício 6.36 para contar o número de respostas corretas e incorretas digitadas pelo aluno. Depois que o aluno digitar 10 respostas, seu programa deve calcular a porcentagem das que estão corretas. Se a porcentagem for menor que 75%, exiba "`Peça ajuda extra ao seu professor.`" e, então, reinicialize o programa para que outro estudante possa tentá-lo. Se a porcentagem for 75% ou maior, exiba "`Parabéns, você está pronto para avançar para o próximo nível!`" e, então, reinicialize o programa para que outro estudante possa tentá-lo.

6.38 **(*Instrução auxiliada por computador: níveis de dificuldade*)** As questões 6.35 a 6.37 desenvolveram um programa de instrução assistida por computador a fim de ajudar a ensinar multiplicação para um aluno do ensino fundamental. Modifique o programa para permitir que o usuário insira um nível de dificuldade. Em um nível de dificuldade 1, o programa deve utilizar apenas números de um único dígito nos problemas; em um nível de dificuldade 2, os números com dois dígitos, e assim por diante.

6.39 **(*Instrução auxiliada por computador: variando os tipos de problema*)** Modifique o programa da Questão 6.38 a fim de permitir ao usuário selecionar um tipo de problema de aritmética a ser estudado. Uma opção de `1` significa apenas problemas de adição, `2` significa apenas problemas de subtração, `3`, de multiplicação, `4`, de divisão e `5`, uma combinação aleatória de problemas de todos esses tipos.