Método `static` `showInputDialog` da classe `JOptionPane`

A linha 10 usa o método **`showInputDialog`** de `JOptionPane` para exibir uma caixa de diálogo de entrada que contém um prompt e um *campo* (conhecido como **campo de texto**) no qual o usuário pode inserir o texto. O argumento do método `showInputDialog` é o prompt que indica o nome que o usuário deve inserir. Ele digita caracteres no campo de texto, depois clica no botão OK ou pressiona a tecla *Enter* para retornar a `String` para o programa. O método `showInputDialog` retorna uma `String` contendo os caracteres digitados pelo usuário. Armazenamos a `String` na variável `name`. Se você pressionar o botão do diálogo Cancel ou a tecla *Esc*, o método retornará `null` e o programa exibirá a palavra "null" como nome.

Método `static` `format` da classe `String`

As linhas 13 e 14 utilizam o método `static String` **`format`** para retornar uma `String` que contém uma saudação com o nome do usuário. O método `format` funciona como `System.out.printf`, exceto que `format` retorna a `String` formatada em vez de exibi-la em uma janela de comando. A linha 17 mostra a saudação em uma caixa de diálogo de mensagem, assim como fizemos na Figura 3.12.

Exercício do estudo de caso GUI e imagens gráficas

3.1 Modifique o programa de adição na Figura 2.7 para utilizar entrada e saída baseadas em caixas de diálogo com os métodos da classe `JOptionPane`. Uma vez que o método `showInputDialog` retorna uma `String`, você deve converter a `String` que o usuário insere em um `int` para utilização em cálculos. O método `static parseInt` da classe `Integer` (pacote `java.lang`) recebe um argumento `String` que representa um inteiro e retorna o valor como um `int`. Se a `String` não contiver um inteiro válido, o programa terminará com um erro.

3.7 Conclusão

Neste capítulo, você aprendeu a criar suas próprias classes e métodos, criar objetos dessas classes e chamar métodos desses objetos para executar ações úteis. Você declarou variáveis de instância de uma classe a fim de manter os dados para cada objeto da classe e também seus próprios métodos para operar nesses dados. Aprendeu a chamar um método para instruí-lo a fazer sua tarefa e de que maneira passar informações para um método como argumentos cujos valores são atribuídos aos parâmetros dele. Descobriu a diferença entre variável local de um método e variável de instância de uma classe, e que apenas variáveis de instância são inicializadas automaticamente. Ainda, verificou como utilizar um construtor da classe para especificar os valores iniciais às variáveis de instância de um objeto. Você viu como criar diagramas de classe UML que modelam os métodos, atributos e construtores das classes. Por fim, você aprendeu a lidar com os números de ponto flutuante (números com separador decimal) — como armazená-los com variáveis de tipo primitivo `double`, como inseri-los com um objeto `Scanner` e como formatá-los com `printf` e o especificador `%f` para propósitos de exibição. [No Capítulo 8, começaremos representando valores monetários precisamente com a classe `BigDecimal`.] Também deve ter começado o estudo de caso opcional de GUI e imagens gráficas, aprendendo a escrever seus primeiros aplicativos GUI. No próximo capítulo iniciaremos nossa introdução às instruções de controle, que especificam a ordem em que as ações de um programa são realizadas. Você as utilizará em seus métodos para especificar como eles devem realizar suas tarefas.

Resumo

Seção 3.2 Variáveis de instância, métodos set *e métodos* get

- Cada classe que você cria torna-se um novo tipo que pode ser utilizado para declarar variáveis e elaborar objetos.
- Você pode declarar novos tipos de classe conforme necessário; essa é uma razão pela qual o Java é conhecido como uma linguagem extensível.

Seção 3.2.1 Classe `Account` com uma variável de instância, um método set *e um método* get

- Toda declaração de classe que inicia com o modificador de acesso `public` deve ser armazenada em um arquivo que tem o mesmo nome que a classe e termina com a extensão de arquivo `.java`.
- Cada declaração de classe contém a palavra-chave `class` seguida imediatamente do nome da classe.
- Os nomes de classe, método e variável são identificadores. Por convenção, todos usam nomes na notação camelo. Os nomes de classe começam com letra maiúscula, e os de método e variável, com uma letra minúscula.
- Um objeto tem atributos que são implementados como variáveis de instância que eles mantêm ao longo de sua vida.
- Existem variáveis de instância antes que os métodos sejam chamados em um objeto, enquanto os métodos são executados e depois que essa ação foi concluída.
- Uma classe normalmente contém um ou mais dos métodos que manipulam as variáveis de instância que pertencem a objetos específicos da classe.
- Variáveis de instância são declaradas dentro de uma declaração de classe, mas fora do corpo das instruções de método da classe.

- Cada objeto (instância) da classe tem sua própria cópia de cada uma das variáveis de instância da classe.
- A maioria das declarações de variável de instância é precedida pela palavra-chave `private`, que é um modificador de acesso. As variáveis ou métodos declarados com o modificador de acesso `private` só são acessíveis a métodos da classe em que são declarados.
- Os parâmetros são declarados em uma lista de itens separados por vírgula, que está localizada entre os parênteses que vêm depois do nome do método na declaração dele. Múltiplos parâmetros são separados por vírgulas. Cada parâmetro deve especificar um tipo seguido por um nome de variável.
- As variáveis declaradas no corpo de um método particular são conhecidas como locais e só podem ser utilizadas nesse método. Quando ele terminar, os valores de suas variáveis locais são perdidos. Os parâmetros de um método são variáveis locais dele.
- O corpo de todos os métodos é delimitado pelas chaves esquerda e direita (`{` e `}`).
- O corpo de cada método contém uma ou mais instruções que executam a(s) tarefa(s) desse método.
- O tipo de retorno do método especifica o tipo de dado retornado para o chamador de um método. A palavra-chave `void` indica que um método realizará uma tarefa, mas não retornará nenhuma informação.
- Os parênteses vazios que seguem um nome de método indicam que ele não requer nenhum parâmetro para realizar sua tarefa.
- Quando um método que especifica um tipo de retorno diferente de `void` for chamado e completar sua tarefa, ele retornará um resultado para seu método de chamada.
- A instrução `return` passa um valor a partir de um método chamado de volta para seu chamador.
- As classes costumam fornecer métodos `public` para permitir aos clientes da classe *set* (configurar) ou *get* (obter) variáveis de instância `private`. Os nomes desses métodos não precisam começar com *set* ou *get,* mas essa convenção de nomenclatura é recomendada.

Seção 3.2.2 Classe `AccountTest` que cria e usa um objeto da classe `Account`

- Uma classe que cria um objeto de outra classe, e então chama os métodos do objeto, é uma *driver*.
- O método `Scanner` `nextLine` lê os caracteres até um caractere de nova linha ser encontrado, depois retorna os caracteres como um método `String`.
- O método `Scanner` `next` lê os caracteres até qualquer um de espaço em branco ser encontrado, então retorna os caracteres como uma `String`.
- A expressão de criação de instância de classe começa com a palavra-chave `new` e estabelece um novo objeto.
- Um construtor é semelhante a um método, mas é chamado implicitamente pelo operador `new` para inicializar as variáveis de instância de um objeto no momento em que ele é criado.
- Para chamar um método de um objeto, o nome da variável deve ser seguido de um ponto separador, do nome de método e de um conjunto de parênteses que contém os argumentos do método.
- Variáveis locais não são inicializadas automaticamente. Cada variável de instância tem um valor inicial padrão — fornecido pelo Java quando você não especifica o valor inicial dela.
- O valor padrão para uma variável de instância do tipo `String` é `null`.
- Uma chamada de método fornece valores — conhecidos como argumentos — para cada um dos parâmetros dele. O valor de cada argumento é atribuído ao parâmetro correspondente no cabeçalho do método.
- O número de argumentos na chamada de método deve corresponder ao de itens na lista de parâmetros da declaração do método.
- Os tipos de argumento na chamada de método devem ser consistentes com os dos parâmetros correspondentes na declaração do método.

Seção 3.2.3 Compilação e execução de um aplicativo com múltiplas classes

- O comando `javac` pode compilar múltiplas classes ao mesmo tempo. Basta listar os nomes dos arquivos do código-fonte após o comando com cada um deles separado do próximo por um espaço. Se o diretório contendo o aplicativo incluir apenas os arquivos de um único aplicativo, você pode compilar todas as classes com o comando `javac *.java`. O asterisco (`*`) em `*.java` indica que todos os arquivos no diretório atual que têm a extensão de nome de arquivo "`.java`" devem ser compilados.

Seção 3.2.4 Diagrama de classe UML de `Account` com uma variável de instância e os métodos set *e* get

- Na UML, cada classe é modelada em um diagrama de classe como um retângulo com três compartimentos. Aquele na parte superior contém o nome da classe centralizado horizontalmente em negrito. O compartimento do meio exibe os atributos da classe, que correspondem às variáveis de instância em Java. O inferior inclui as operações da classe, que correspondem a métodos e construtores em Java.
- A UML representa variáveis de instância como um nome de atributo, seguido por dois-pontos e o tipo.
- Os atributos privados são precedidos por um sinal de subtração (–) na UML.
- A UML modela operações listando o nome delas seguido por um conjunto de parênteses. Um sinal de adição (+) na frente do nome da operação indica que é uma do tipo public na UML (isto é, um método `public` em Java).
- A UML modela um parâmetro de uma operação listando o nome dele, seguido por um caractere de dois-pontos e o tipo dele entre parênteses depois do nome de operação.
- A UML indica o tipo de retorno de uma operação colocando dois-pontos e ele depois dos parênteses que se seguem ao nome da operação.

- Os diagramas de classe UML não especificam tipos de retorno para operações que não retornam valores.
- Declarar variáveis de instância `private` é conhecido como ocultar dados ou informações.

Seção 3.2.5 Notas adicionais sobre a classe `AccountTest`

- Você deve chamar a maioria dos métodos, exceto `main`, explicitamente para instruí-los a fazer suas tarefas.
- Uma parte fundamental da ativação da JVM para localizar e chamar o método `main` a fim de começar a execução do aplicativo é a palavra-chave `static`, que indica que `main` é um método `static` que pode ser chamado sem antes criar um objeto da classe em que esse método é declarado.
- A maioria das outras classes que você utilizará nos programas Java precisa ser importada explicitamente. Há um relacionamento especial entre as classes que são compiladas no mesmo diretório. Por padrão, essas classes são consideradas como estando no mesmo pacote — conhecido como pacote padrão. As classes do mesmo pacote são importadas implicitamente para os arquivos de código-fonte de outras classes desse mesmo pacote. Uma declaração `import` não é necessária quando uma classe em um pacote utiliza outra no mesmo pacote.
- Uma declaração `import` não é necessária se você sempre se referir a uma classe com um nome totalmente qualificado, que inclui o nome do pacote e o nome da classe.

Seção 3.2.6 Engenharia de software com variáveis de instância `private` e métodos set *e* get *`public`*

- Declarar variáveis de instância `private` é conhecido como ocultar dados ou informações.

Seção 3.3 Tipos primitivos versus *tipos por referência*

- Tipos no Java são divididos em duas categorias — primitivos e por referência. Os tipos primitivos são `boolean`, `byte`, `char`, `short`, `int`, `long`, `float` e `double`. Todos os outros são por referência; portanto, classes, que especificam os tipos de objeto, são tipos por referência.
- Uma variável de tipo primitivo pode armazenar exatamente um valor de seu tipo declarado por vez.
- As variáveis de instância de tipo primitivo são inicializadas por padrão. Variáveis dos tipos `byte`, `char`, `short`, `int`, `long`, `float` e `double` são inicializadas como `0`. As variáveis de tipo `boolean` são inicializadas como `false`.
- Variáveis de tipo por referência (chamadas referências) armazenam o local de um objeto na memória do computador. Essas variáveis referenciam objetos no programa. O objeto que é referenciado pode conter muitas variáveis de instância e métodos.
- As variáveis de instância de tipo por referência são inicializadas por padrão como valor `null`.
- Uma referência a um objeto é necessária para chamar os métodos de um objeto. Uma variável de tipo primitivo não referencia um objeto e, portanto, não pode ser utilizada para invocar um método.

Seção 3.4 Classe `Account`: inicialização de objetos com construtores

- Cada classe que você declara pode fornecer um construtor com parâmetros a ser utilizados para inicializar um objeto de uma classe quando ele for criado.
- O Java requer uma chamada de construtor para cada objeto que é criado.
- Construtores podem especificar parâmetros, mas não tipos de retorno.
- Se uma classe não definir construtores, o compilador fornecerá um construtor padrão sem parâmetros, e as variáveis de instância da classe serão inicializadas com seus valores padrão.
- Se você declarar um construtor para uma classe, o compilador *não* criará um *construtor padrão* para ela.
- A UML modela os construtores no terceiro compartimento de um diagrama de classe. Para distinguir entre um construtor e operações de uma classe, a UML coloca a palavra "constructor" entre aspas francesas (« e ») antes do nome do construtor.

Seção 3.5 A classe `Account` com um saldo; números de ponto flutuante

- Um número de ponto flutuante tem um ponto decimal. O Java fornece dois tipos primitivos para armazenar números de ponto flutuante na memória — `float` e `double`.
- As variáveis de tipo `float` representam números de ponto flutuante de precisão simples e têm sete dígitos significativos. Já as variáveis de tipo `double` indicam números de ponto flutuante de dupla precisão. Elas requerem duas vezes a quantidade de memória das variáveis `float` e fornecem 15 dígitos significativos — aproximadamente o dobro da precisão de variáveis `float`.
- Literais de ponto flutuante são do tipo `double` por padrão.
- O método `nextDouble` de `Scanner` retorna um valor `double`.
- O especificador de formato `%f` é utilizado para gerar saída de valores de tipo `float` ou `double`. Já o especificador de formato `%.2f` especifica que dois dígitos da precisão devem ser gerados à direita do ponto decimal no número de ponto flutuante.
- O valor padrão para uma variável de instância do tipo `double` é `0.0`, e o valor padrão para uma variável de instância do tipo `int` é `0`.

Exercícios de revisão

3.1 Preencha as lacunas em cada uma das seguintes sentenças:

a) Toda declaração de classe que inicia com a palavra-chave _____ deve ser armazenada em um arquivo que tem exatamente o mesmo nome que a classe e terminar com a extensão de nome do arquivo `.java`.

b) A palavra-chave _____ em uma declaração de classe é imediatamente seguida pelo nome da classe.

c) A palavra-chave _____ solicita memória do sistema para armazenar um objeto, e então chama o construtor da classe correspondente para inicializar esse objeto.

d) Todo parâmetro deve especificar um(a) _____ e um(a) _____.

e) Por padrão, as classes que são compiladas no mesmo diretório são consideradas como estando no mesmo pacote, conhecido como _____.

f) O Java fornece dois tipos primitivos para armazenar números de ponto flutuante na memória: _____ e _____.

g) As variáveis de tipo `double` representam números de ponto flutuante de _____.

h) O método `scanner` _____ retorna um valor `double`.

i) A palavra-chave `public` é um _____ de acesso.

j) O tipo de retorno _____ indica que um método não retornará um valor.

k) O método `Scanner` _____ lê os caracteres até encontrar um caractere de nova linha, então retorna esses caracteres como uma `String`.

l) A classe `String` está no pacote _____.

m) Um(a) _____ não é requerido(a) se você sempre referenciar uma classe por meio do seu nome completamente qualificado.

n) Um(a) _____ é um número com um ponto de fração decimal, como 7,33, 0,0975 ou 1000,12345.

o) As variáveis de tipo `float` representam _____ números de ponto flutuante de dupla precisão.

p) O especificador de formato _____ é utilizado para gerar saída de valores de tipo `float` ou `double`.

q) Os tipos no Java são divididos em duas categorias — tipo _____ e tipo _____.

3.2 Determine se cada uma das seguintes sentenças é *verdadeira* ou *falsa*. Se *falsa*, explique por quê.

a) Por convenção, os nomes de método são iniciados com letra maiúscula, e todas as palavras subsequentes a ele também começam com letra maiúscula.

b) Uma declaração `import` não é necessária quando uma classe em um pacote utiliza outra no mesmo pacote.

c) Parênteses vazios que se seguem a um nome de método em uma declaração indicam que ele não requer nenhum parâmetro para realizar sua tarefa.

d) Uma variável de tipo primitivo pode ser utilizada para invocar um método.

e) As variáveis declaradas no corpo de um método particular são conhecidas como variáveis de instância e podem ser utilizadas em todos os métodos da classe.

f) O corpo de todos os métodos é delimitado pelas chaves esquerda e direita (`{` e `}`).

g) As variáveis locais de tipo primitivo são inicializadas por padrão.

h) As variáveis de instância de tipo por referência são inicializadas por padrão com o valor `null`.

i) Qualquer classe que contém `public static void main(String[] args)` pode ser usada para executar um aplicativo.

j) O número de argumentos na chamada de método deve corresponder ao de itens na lista de parâmetros da declaração desse método.

k) Os valores de ponto flutuante que aparecem no código-fonte são conhecidos como literais de ponto flutuante e são tipos `float` por padrão.

3.3 Qual é a diferença entre uma variável local e uma variável de instância?

3.4 Explique o propósito de um parâmetro de método. Qual a diferença entre um parâmetro e um argumento?

Respostas dos exercícios de revisão

3.1 a) `public`. b) `class`. c) `new`. d) tipo, nome. e) pacote padrão. f) `float`, `double`. g) precisão dupla. h) `nextDouble`. i) modificador. j) `void`. k) `nextLine`. l) `java.lang`. m) declaração `import`. n) número de ponto flutuante. o) simples. p) `%f`. q) primitivo, referência.

3.2 a) Falsa. Por convenção, os nomes de método são iniciados com letra minúscula e todas as palavras subsequentes começam com letra maiúscula. b) Verdadeira. c) Verdadeira. d) Falsa. Uma variável de tipo primitivo não pode ser utilizada para invocar um método — uma referência a um objeto é necessária para que os métodos do objeto possam ser invocados. e) Falsa. Essas variáveis são chamadas variáveis locais e só podem ser utilizadas no método em que são declaradas. f) Verdadeira. g) Falsa. As variáveis de instância de tipo primitivo são inicializadas por padrão. Deve-se atribuir um valor explicitamente a cada variável local. h) Verdadeira. i) Verdadeira. j) Verdadeira. k) Falsa. Esses literais são de tipo `double` por padrão.

3.3 Uma variável local é declarada no corpo de um método e só pode ser utilizada do ponto em que isso acontece até o fim da declaração do método. Uma variável de instância é declarada em uma classe, mas não no corpo de qualquer um dos métodos dessa classe. Além disso, as variáveis de instância são acessíveis a todos os métodos da classe. (Veremos uma exceção disso no Capítulo 8.)

3.4 Um parâmetro representa informações adicionais que um método requer para realizar sua tarefa. Cada parâmetro requerido por um método é especificado na declaração do método. Um argumento é o valor real de um parâmetro de método. Quando um método é chamado, os valores de argumento são passados para os parâmetros correspondentes desse método para que ele possa realizar sua tarefa.

Questões

3.5 **(*Palavra-chave* `new`)** Qual é o objetivo da palavra-chave `new`? Explique o que acontece quando você a utiliza.

3.6 **(*Construtores padrão*)** O que é um construtor padrão? Como as variáveis de instância de um objeto são inicializadas se uma classe tiver somente um construtor padrão?

3.7 **(*Variáveis de instância*)** Explique o propósito de uma variável de instância.

3.8 **(*Usando classes sem importá-las*)** A maioria das classes precisa ser importada antes de ser usada em um aplicativo. Por que cada aplicativo pode utilizar as classes `System` e `String` sem importá-las antes?

3.9 **(*Usando uma classe sem importá-la*)** Explique como um programa pode usar a classe `Scanner` sem importá-la.

3.10 **(*Métodos* set *e* get)** Explique por que uma classe pode fornecer um método *set* e um método *get* para uma variável de instância.

3.11 **(*Classe `Account` modificada`*)** Modifique a classe `Account` (Figura 3.8) para fornecer um método chamado `withdraw` que retira dinheiro de uma `Account`. Assegure que o valor de débito não exceda o saldo de `Account`. Se exceder, o saldo deve ser deixado inalterado e o método deve imprimir uma mensagem que indica `"Withdrawal amount exceeded account balance" [Valor de débito excedeu o saldo da conta]`. Modifique a classe `AccountTest` (Figura 3.9) para testar o método `withdraw`.

3.12 **(*Classe `Invoice`*)** Crie uma classe chamada `Invoice` para que uma loja de suprimentos de informática a utilize para representar uma fatura de um item vendido nela. Uma `Invoice` (fatura) deve incluir quatro partes das informações como variáveis de instância — o número (tipo `String`), a descrição (tipo `String`), a quantidade comprada de um item (tipo `int`) e o preço por item (`double`). Sua classe deve ter um construtor que inicializa as quatro variáveis de instância. Forneça um método *set* e um *get* para cada variável de instância. Além disso, forneça um método chamado `getInvoiceAmount` que calcula o valor de fatura (isto é, multiplica a quantidade pelo preço por item) e depois retorna esse valor como `double`. Se a quantidade não for positiva, ela deve ser configurada como `0`. Se o preço por item não for positivo, ele deve ser configurado como `0.0`. Escreva um aplicativo de teste chamado `InvoiceTest` que demonstra as capacidades da classe `Invoice`.

3.13 **(*Classe `Employee`*)** Crie uma classe chamada `Employee` que inclua três variáveis de instância — um primeiro nome (tipo `String`), um sobrenome (tipo `String`) e um salário mensal (`double`). Forneça um construtor que inicializa as três variáveis de instância. Forneça um método *set* e um *get* para cada variável de instância. Se o salário mensal não for positivo, não configure seu valor. Escreva um aplicativo de teste chamado `EmployeeTest` que demonstre as capacidades da classe `Employee`. Crie dois objetos `Employee` e exiba o salário *anual* de cada objeto. Então dê a cada `Employee` um aumento de 10% e exiba novamente o salário anual de cada `Employee`.

3.14 **(*Classe `Date`*)** Crie uma classe chamada `Date` que inclua três variáveis de instância — mês (tipo `int`), dia (tipo `int`) e ano (tipo `int`). Forneça um construtor que inicializa as três variáveis de instância supondo que os valores fornecidos estejam corretos. Ofereça um método *set* e um *get* para cada variável de instância. Apresente um método `displayDate` que exiba mês, dia e ano separados por barras normais (`/`). Escreva um aplicativo de teste chamado `DateTest` que demonstre as capacidades da classe `Date`.

3.15 **(*Removendo código duplicado no método* `main`)** Na classe `AccountTest` da Figura 3.9, o método `main` contém seis instruções (linhas 13 e 14, 15 e 16, 28 e 29, 30 e 31, 40 e 41 e 42 e 43) e cada uma exibe `name` e `balance` do objeto `Account`. Estude essas instruções e você perceberá que elas só diferem no objeto `Account` sendo manipulado — `account1` ou `account2`. Neste exercício, você definirá um novo método `displayAccount` que contém *uma* cópia dessa instrução de saída. O parâmetro do método será um objeto `Account` e o método irá gerar `name` e `balance` dele. Então você substituirá as seis instruções duplicadas em `main` por chamadas para `displayAccount` passando como argumento o objeto específico `Account` para saída.

Modifique a classe `AccountTest` da Figura 3.9 para declarar o seguinte método `displayAccount` *após* a chave direita de fechamento de `main` e *antes* da chave direita de fechamento da classe `AccountTest`:

```
public static void displayAccount(Account accountToDisplay)
{
   // coloque aqui a instrução que exibe
   // o name e o balance de accountToDisplay
}
```

Substitua o comentário no corpo do método por uma instrução que exiba `name` e `balance` de `accountToDisplay`.

Lembre-se de que `main` é um método `static`, assim pode ser chamado sem antes criar um objeto da classe em que é declarado. Também declaramos o método `displayAccount` como um método `static`. Quando `main` tem de chamar outro método na mesma classe sem antes criar um objeto dela, o outro método *também* deve ser declarado `static`.

Depois de concluir a declaração de `displayAccount`, modifique `main` para substituir as instruções que exibem `name` e `balance` de cada `Account` pelas chamadas para `displayAccount` — cada uma recebendo como seu argumento o objeto `account1` ou `account2`, como apropriado. Então, teste a classe `AccountTest` atualizada para garantir que ela produz a mesma saída como mostrado na Figura 3.9.

Fazendo a diferença

3.16 **(*Calculadora de frequência cardíaca alvo*)** Ao fazer exercícios físicos, você pode utilizar um monitor de frequência cardíaca para ver se sua frequência permanece dentro de um intervalo seguro sugerido pelos seus treinadores e médicos. Segundo a American Heart Association (AHA) (`www.americanheart.org/presenter.jhtml?identifier=4736`), a fórmula para calcular a *frequência cardíaca máxima* por minuto é 220 menos a idade em anos. Sua *frequência cardíaca alvo* é um intervalo entre 50-85% da sua frequência cardíaca máxima. [***Observação:*** essas fórmulas são estimativas fornecidas pela AHA. As frequências cardíacas máximas e alvo podem variar com base na saúde, capacidade física e sexo da pessoa. **Sempre consulte um médico ou profissional de saúde qualificado antes de começar ou modificar um programa de exercícios físicos.**] Crie uma classe chamada `HeartRates`. Os atributos da classe devem incluir o nome, sobrenome e data de nascimento da pessoa (consistindo em atributos separados para mês, dia e ano de nascimento). Sua classe deve ter um construtor que receba esses dados como parâmetros. Para cada atributo forneça métodos *set* e *get*. A classe também deve incluir um método que calcule e retorne a idade (em anos), um que calcule e retorne a frequência cardíaca máxima e um que calcule e retorne a frequência cardíaca alvo da pessoa. Escreva um aplicativo Java que solicite as informações da pessoa, instancie um objeto da classe `HeartRates` e imprima as informações a partir desse objeto — incluindo nome, sobrenome e data de nascimento da pessoa — calcule e imprima a idade da pessoa (em anos), seu intervalo de frequência cardíaca máxima e sua frequência cardíaca alvo.

3.17 **(*Computadorização dos registros de saúde*)** Uma questão relacionada à assistência médica discutida ultimamente nos veículos de comunicação é a computadorização dos registros de saúde. Essa possibilidade está sendo abordada de maneira cautelosa por causa de preocupações quanto à privacidade e à segurança de dados sigilosos, entre outros motivos. [Iremos discutir essas preocupações em exercícios posteriores.] A computadorização dos registros de saúde pode facilitar que pacientes compartilhem seus perfis e históricos de saúde entre vários profissionais de saúde. Isso talvez aprimore a qualidade da assistência médica, ajude a evitar conflitos e prescrições erradas de medicamentos, reduza custos em ambulatórios e salve vidas. Neste exercício, você projetará uma classe `HealthProfile` "inicial" para uma pessoa. Os atributos da classe devem incluir nome, sobrenome, sexo, data de nascimento (consistindo em atributos separados para mês, dia e ano de nascimento), altura (em metros) e peso (em quilogramas) da pessoa. Sua classe deve ter um construtor que receba esses dados. Para cada atributo, forneça métodos *set* e *get*. A classe também deve incluir métodos que calculem e retornem a idade do usuário em anos, intervalo de frequência cardíaca máxima e frequência cardíaca alvo (veja o Exercício 3.16), além de índice de massa corporal (IMC; veja o Exercício 2.33). Escreva um aplicativo Java que solicite as informações da pessoa, instancie um objeto da classe `HealthProfile` para ela e imprima as informações a partir desse objeto — incluindo nome, sobrenome, sexo, data de nascimento, altura e peso da pessoa —, e então calcule e imprima a idade em anos, IMC, intervalo de frequência cardíaca máxima e frequência cardíaca alvo. Ele também deve exibir o gráfico de valores IMC do Exercício 2.33.