3

Introdução a classes e objetos

OBJETIVOS

- Neste capítulo, você aprenderá:
- O que são classes, objetos, métodos e variáveis de instância.
- Como declarar uma classe e utilizá-la para criar um objeto.
- Como declarar métodos em uma classe para implementar os comportamentos da classe.
- Como declarar variáveis de instância em uma classe para implementar os atributos da classe.
- Como chamar o método de um objeto para fazer esse método realizar sua tarefa.
- As diferenças entre variáveis de instância de uma classe e variáveis locais de um método.
- Como utilizar um construtor para assegurar que os dados de um objeto sejam inicializados quando o objeto for criado.
- As diferenças entre tipos por referência primitivos.

3.1 Introdução

- **Classes**
- **Números de ponto flutuante**

3.2 Classes, objetos, métodos e variáveis de instância

- **Classes fornecem um ou mais métodos.**
- **Métodos representam tarefas em um programa:**
  - **O método descreve os mecanismos que realmente realizam suas tarefas.**
  - **Oculta do usuário as tarefas complexas que ele realiza.**
  - **Chamada de método instrui o método a realizar sua tarefa.**

3.2 Classes, objetos, métodos e variáveis de instância (*Continuação*)

- **Classes contêm um ou mais atributos:**
  - **Especificados pelas variáveis de instância.**
  - **Transportados com o objeto à medida que são utilizados.**

3.3 Declarando uma classe com um método e instanciando um objeto de uma classe

- Cada declaração de classe que inicia com a palavra-chave `public` deve ser armazenada em um arquivo que tenha o mesmo nome da classe e terminar com a extensão de nome do arquivo `.java`.

Classe GradeBook

- **A palavra-chave `public` é um modificador de acesso.**
- **Declarações de classe incluem:**
  - **Modificador de acesso.**
  - **Palavra-chave `class`.**
  - **Par de chaves esquerda e direita.**

Class GradeBook

- **Declarações de método:**
  - **Palavra-chave `public` indica o método disponível ao público.**
  - **Palavra-chave `void` indica nenhum tipo de retorno.**
  - **Modificador de acesso, tipo de retorno, nome do método e parênteses compõem o cabeçalho do método.**

Erro comum de programação 3.1

Declarar mais de uma classe `public` no mesmo arquivo é um erro de compilação.

Resumo

GradeBook.jav

```
1  // Fig. 3.1: GradeBook.java
2  // Declaração de classe com um método.
3
4  public class GradeBook
5  {
6     // exibe uma mensagem de boas-vindas para o usuário GradeBook
7     public void displayMessage()
8     {
9        System.out.println( "Welcome to the Grade Book!" );
10    } // fim do método displayMessage
11
12 } // fim da classe GradeBook
```

Imprime linha do texto na saída

Classe GradeBookTest

- **O Java é extensível:**
  - **Os programadores podem criar novas classes.**
- **Expressão de criação de instância de classe:**
  - **Palavra-chave** `new`.
  - **Então, nome da classe a criar e os parênteses.**
- **Chamando um método:**
  - **Nome de objeto, daí o separador ponto (.).**
  - **Depois, o nome e os parênteses do método.**

Resumo

GradeBookTest.Java

```
1  // Fig. 3.2: GradeBookTest.java
2  // Cria um objeto GradeBook e chama seu método displayMessage.
3
4  public class GradeBookTest
5  {
6     // método main inicia a execução de programa
7     public static void main( String args[] )
8     {
9        // cria um objeto GradeBook  e o atribui a my
10       GradeBook myGradeBook = new GradeBook();
11
12       // chama método displayMessage de myGradeBook
13       myGradeBook.displayMessage();
14    } // fim de main
15
16 } // fim da classe GradeBookTest
```

```
Welcome to the Grade Book!
```

Utiliza a expressão de criação de instância de classe para criar o objeto da classe `GradeBook`

Chama o método `displayMessage` utilizando o objeto `GradeBook`

Compilando um aplicativo com múltiplas classes

- **Compilando múltiplas classes:**
  - **Lista cada arquivo `.java` separadamente, divididos por espaços.**
  - **Compila com `*.java` para compilar todos os arquivos `.java` nesse diretório.**

Diagrama de classe UML para a classe `GradeBook`

- **Diagramas de classe UML:**
  - **Compartimento superior contém o nome da classe.**
  - **Compartimento intermediário contém atributos ou variáveis de instância da classe.**
  - **Compartimento inferior contém operações ou métodos da classe.**
    - **Sinal de adição indica os métodos `public`.**

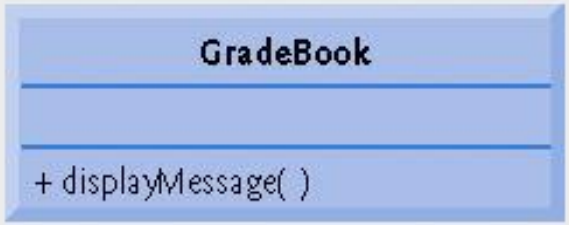

Figura 3.3 | Diagrama de classe UML indicando que a classe `GradeBook` tem uma operação `public displayMessage`.

3.4 Declarando um método com um parâmetro

- **Parâmetros de método:**
  - **Informações adicionais passadas para um método.**
  - **Informações adicionais fornecidas na chamada de método com argumentos.**

3.4 Declarando um método com um parâmetro (*Continuação)*

- **Métodos** `Scanner`:
  - `nextLine` **lê a próxima linha da entrada.**
  - `next` **lê a próxima palavra da entrada.**

Resumo

GradeBook.java

```
1  // Fig. 3.4: GradeBook.java
2  // Declaração de classe com um método que tem um parâmetro.
3
4  public class GradeBook
5  {
6     // exibe uma mensagem de boas-vindas para o usuário GradeBook
7     public void displayMessage( String courseName )
8     {
9        System.out.printf( "Welcome to the grade book for\n%s!\n",
10          courseName );
11    } // fim do método displayMessage
12
13 } // fim da classe GradeBook
```

Chama o método `printf` com o argumento `courseName`

Resumo

GradeB \
ookTes
t.Java

```
1  // Fig. 3.5: GradeBookTest.java
2  // Create GradeBook object and pass a String to
3  // its displayMessage method.
4  import java.util.Scanner; // program uses Scanner
5
6  public class GradeBookTest
7  {
8     // main method begins program execution
9     public static void main( String args[] )
10    {
11       // create Scanner to obtain input from command window
12       Scanner input = new Scanner( System.in );
13
14       // create a GradeBook object and assign it
15       GradeBook myGradeBook = new GradeBook();
16
17       // prompt for and input course name
18       System.out.println( "Please enter the course name:" );
19       String nameOfCourse = input.nextLine(); //
20       System.out.println(); // outputs a blank l
21
22       // call myGradeBook's displayMessage method
23       // and pass nameOfCourse as an argument
24       myGradeBook.displayMessage( nameOfCourse );
25    } // end main
26
27 } // end class GradeBookTest
```

Chama o método `nextLine` para ler uma linha de entrada

Chama `displayMessage` com um argumento

```
Please enter the course name:
CS101 Introduction to Java Programming

Welcome to the grade book for
CS101 Introduction to Java Programming!
```

Observação de engenharia de software 3.1

Normalmente, os objetos são criados com `new`. Uma exceção é um literal de string que está entre aspas, como "`hello`".

Os literais de string são referências a objetos `String` que são criados implicitamente pelo Java.

Mais sobre argumentos e parâmetros

- **Parâmetros especificados na lista de parâmetros de um método:**
  - **Parte do cabeçalho do método.**
  - **Utiliza uma lista separada por vírgulas.**

Erro comum de programação 3.2

Ocorrerá um erro de compilação se o número de argumentos em uma chamada de método não corresponder ao número de parâmetros na declaração de método.

Erro comum de programação 3.3

Ocorrerá um erro de compilação se os tipos dos argumentos em uma chamada de método não forem consistentes com os tipos dos parâmetros correspondentes na declaração do método.

Diagrama de classe UML para a classe `GradeBook`

- **Diagrama de classe UML:**
  - **Parâmetros especificados por nome de parâmetro seguido por dois-pontos e pelo tipo de parâmetro nos parênteses que seguem o nome da operação.**

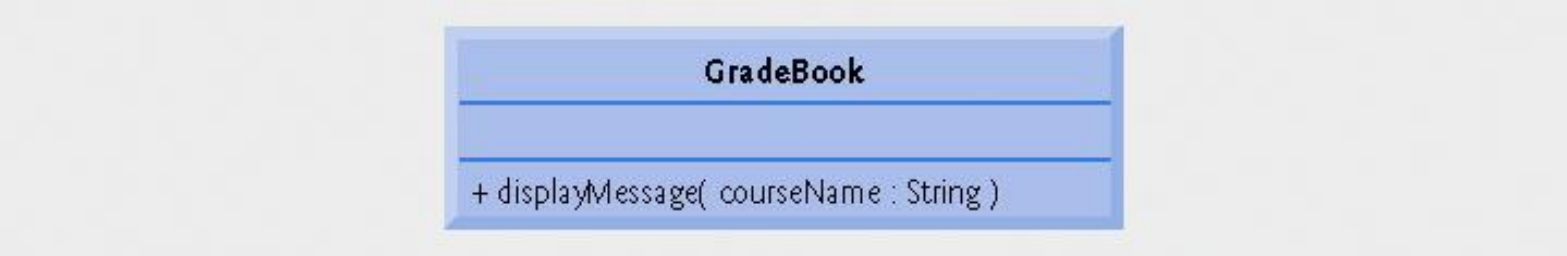

Figura 3.6 | Diagrama de classe UML indicando que a classe `GradeBook` tem uma operação `displayMessage` com um parâmetro `courseName` do tipo UML `String`.

Notas sobre as declarações `Import`

- `java.lang` **é implicitamente importada em todo programa.**
- **Pacote-padrão:**
  - **Contém classes compiladas no mesmo diretório.**
  - **Implicitamente importada para o código-fonte dos outros arquivos no diretório.**
- **Empacota nomes desnecessários caso sejam utilizados nomes completamente qualificados.**

Observação de engenharia de software 3.2

O compilador Java não exigirá as declarações `import` em um arquivo de código-fonte Java caso o nome de classe completamente qualificado seja especificado toda vez que um nome de classe for utilizado no código-fonte.

Mas a maioria dos programadores em Java considera incômoda a utilização dos nomes completamente qualificados e, em vez disso, eles preferem utilizar declarações `import`.

3.5 Variáveis de instância, métodos *set* e *get*

- **Variáveis declaradas no corpo do método:**
  - **Conhecidas como *variáveis locais*.**
  - **Somente podem ser utilizadas nesse método.**
- **Variáveis declaradas em uma declaração de classe:**
  - **Chamadas de *campos* ou *variáveis de instância*.**
  - **Cada objeto (instância) da classe tem uma instância separada da variável.**

Resumo

GradeBook.java

```
1  // Fig. 3.7: GradeBook.java
2  // Classe GradeBook que contém uma variável de instância courseName
3  // e métodos para configurar e obter seu valor.
4
5  public class GradeBook
6  {
7     private String courseName; // nome do curos para este GradeBook
8
9     // método para configurar o nome do curso
10    public void setCourseName( String name )
11    {
12       courseName = name; // armazena o nome do curso
13    } // fim do método setCourseName
14
15    // método para recuperar o nome do curso
16    public String getCourseName()
17    {
18       return courseName;
19    } // fim do método getCourseName
20
21    // exibe uma mensagem de boas-vindas para o usuário GradeBook
22    public void displayMessage()
23    {
24       // essa instrução chama getCourseName para obter o
25       // nome do curso que esse GradeBook representa
26       System.out.printf( "Welcome to the grade book for\n%s!\n",
27          getCourseName() );
28    } // fim do método displayMessage
29
30 } // fim da classe GradeBook
```

Variável de instância courseName

Método *set* para courseName

Método *get* para courseName

Chama o método *get*

Modificadores de acesso `public` e `private`

- **Palavra-chave `private`:**
  - **Utilizada na maioria das variáveis de instância.**
  - **Variáveis e métodos `private` são acessíveis somente aos métodos da classe em que eles são declarados.**
  - **Declarar variáveis de instância como `private` é conhecido como *ocultamento de dados*.**
- **Tipo de retorno:**
  - **Indica itens retornados pelo método.**
  - **Declarado no cabeçalho do método.**

Observação de engenharia de software 3.3

Anteceda cada campo e declaração de método com um modificador de acesso.

Como regra geral, as variáveis de instância devem ser declaradas `private` e os métodos devem ser declarados `public`. (Veremos que é apropriado declarar certos métodos `private`, se eles forem acessados apenas por outros métodos da classe.)

Boa prática de programação 3.1

Preferimos listar os campos de uma classe primeiro para que, ao ler o código, você veja os nomes e tipos das variáveis antes de colocá-los em uso nos métodos da classe.

É possível listar os campos da classe em qualquer lugar na classe fora de suas declarações de método, mas sua dispersão tende a resultar em um código de difícil leitura.

Boa prática de programação 3.2

Coloque uma linha em branco entre as declarações de método para separar os métodos e aprimorar a legibilidade do programa.

Classe `GradeBookTest` que demonstra a classe GradeBook

- **Valor inicial padrão:**
  - **Fornecido a todos os campos não-inicializados.**
  - **Igual a `null` para `String`s.**

Métodos *set* e *get*

- **Variáveis de instância `private`:**
  - **Não podem ser acessadas diretamente pelos clientes do objeto.**
  - **Utilize métodos *set* para alterar o valor.**
  - **Utilize métodos *get* para recuperar o valor.**

Resumo

GradeBookTest.Java

(1 de 2)

```
1  // Fig. 3.8: GradeBookTest.java
2  // Cria e manipula um objeto GradeBook.
3  import java.util.Scanner; // programa utiliza Scanner
4
5  public class GradeBookTest
6  {
7     // método main inicia a execução de programa
8     public static void main( String args[] )
9     {
10       // cria Scanner para obter entrada a partir da janela de comando
11       Scanner input = new Scanner( System.in );
12
13       // cria um objeto GradeBook e o atribui a myGradeBook
14       GradeBook myGradeBook = new GradeBook();
15
16       // exibe valor inicial de courseName
17       System.out.printf( "Initial course name is: %s\n\n",
18          myGradeBook.getCourseName() );
19
```

Chame o método *get* para `courseName`

Resumo

```
20      // solicita e lê o nome do curso
21      System.out.println( "Please enter the course name:" );
22      String theName = input.nextLine(); // lê uma linha de texto
23      myGradeBook.setCourseName( theName ); // c
24      System.out.println(); // gera saída de uma
25
26      // exibe mensagem de boas-vindas depois de especificar nome do curso
27      myGradeBook.displayMessage();
28   } // fim de main
29
30 } // fim da classe GradeBookTest
```

Chama o método *set* para `courseName`

Chama `displayMessage`

GradeBookTest.Java

(2 de 2)

```
Initial course name is: null

Please enter the course name:
CS101 Introduction to Java Programming

Welcome to the grade book for
CS101 Introduction to Java Programming!
```

Diagrama de classe UML de `GradeBook` com uma variável de instância e os métodos *set* e *get*

- **Atributos:**
  - **Listado no compartimento intermediário da classe.**
  - **Nome do atributo seguido por um caractere de dois-pontos e o tipo de atributo.**
- **Tipo de retorno de um método:**
  - **Indicado com um dois-pontos e tipo de retorno após os parênteses, que se seguem ao nome da operação.**

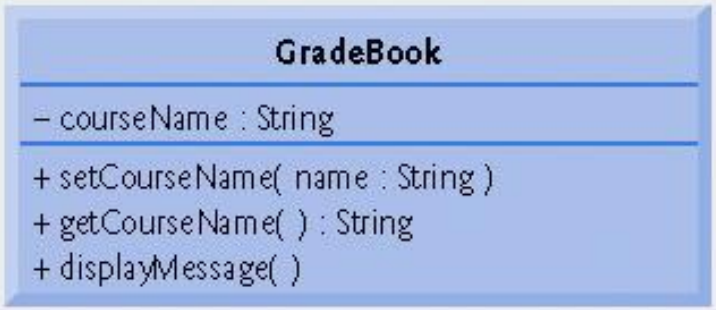

Figura 3.9 | Diagrama de classe UML indicando que a classe `GradeBook` tem um atributo `courseName` de tipo UML `String` e três operações `setCourseName` (com um parâmetro de nome de tipo UML `String`), `getCourseName` (retorna o tipo UML `String`) e displayMessage.

Tipos primitivos *versus* tipos por referência

- **Tipos no Java:**
  - **Primitivos:**
    - `boolean`, `byte`, `char`, `short`, `int`, `long`, `float`, `double`.
  - **Referência (às vezes chamada *tipos não-primitivos*):**
    - **Objetos.**
    - **Valor inicial padrão de `null`.**
    - **Utilizado para invocar os métodos de um objeto.**

Observação de engenharia de software 3.4

O tipo declarado de uma variável (por exemplo, `int`, `double` ou `GradeBook`) indica se a variável é de um tipo primitivo ou tipo por referência.

Se o tipo de uma variável não for um dos oito tipos primitivos, então ele é um tipo por referência. (Por exemplo, `Account account1` indica que `account1` é uma referência a um objeto `Account`.)

3.7 Inicializando objetos com construtores

- **Construtores:**
  - **Inicializam um objeto de uma classe.**
  - **O Java requer um construtor para cada classe.**
  - **O Java fornecerá um construtor sem argumentos-padrão, caso nenhum seja fornecido.**
  - **Chamados quando a palavra-chave `new` é precedida pelo nome e parênteses da classe.**

Resumo

GradeBook.java

(1 de 2)

```
1  // Fig. 3.10: GradeBook.java
2  // Classe GradeBook com um construtor para inicializar o nome de um curso.
3
4  public class GradeBook
5  {
6     private String courseName; // nome do curso para esse GradeBook
7
8     // construtor inicializa courseName com String fornecido como argumento
9     public GradeBook( String name )
10    {
11       courseName = name; // inicializa courseName
12    } // fim do construtor
13
14    // método para configurar o nome do curso
15    public void setCourseName( String name )
16    {
17       courseName = name; // armazena o nome do curso
18    } // fim do método setCourseName
19
20    // método para recuperar o nome do curso
21    public String getCourseName()
22    {
23       return courseName;
24    } // fim do método getCourseName
```

Constructor to initialize `courseName` variable

```
25
26    // exibe uma mensagem de boas-vindas para o usuário GradeBook
27    public void displayMessage()
28    {
29       // essa instrução chama getCourseName para obter o
30       // nome do curso que esse GradeBook representa
31       System.out.printf( "Welcome to the grade book for\n%s!\n",
32          getCourseName() );
33    } // fim do método displayMessage
34
35 } // fim da classe GradeBook
```

Resumo

GradeBook.java

(2 de 2)

Resumo

GradeBookTest.Java

```
1  // Fig. 3.11: GradeBookTest.java
2  // GradeBook constructor used to specify the course name at the
3  // time each GradeBook object is created.
4
5  public class GradeBookTest
6  {
7     // main method begins program execution
8     public static void main( String args[] )
9     {
10       // create GradeBook object
11       GradeBook gradeBook1 = new GradeBook(
12          "CS101 Introduction to Java Programming" );
13       GradeBook gradeBook2 = new GradeBook(
14          "CS102 Data Structures in Java" );
15
16       // display initial value of courseName for each GradeBook
17       System.out.printf( "gradeBook1 course name is: %s\n",
18          gradeBook1.getCourseName() );
19       System.out.printf( "gradeBook2 course name is: %s\n",
20          gradeBook2.getCourseName() );
21    } // end main
22
23 } // end class GradeBookTest
```

Chama o construtor para criar o primeiro objeto GradeBook

Cria o segundo objeto GradeBook

```
gradeBook1 course name is: CS101 Introduction to Java Programming
gradeBook2 course name is: CS102 Data Structures in Java
```

Dica de prevenção de erro 3.1

A menos que a inicialização-padrão de variáveis de instância de sua classe seja aceitável, forneça um construtor para assegurar que as variáveis de instância da sua classe sejam adequadamente inicializadas com valores significativos quando cada novo objeto de sua classe for criado.

Adicionando o construtor ao diagrama de classe UML da classe `GradeBookTest`

- **Diagrama de classe UML:**
  - **A UML modela construtores no terceiro compartimento de uma classe.**
  - **A UML coloca a palavra '<<constructor>>' antes do nome do construtor.**
  - **É habitual listar construtores antes de outras operações no terceiro compartimento.**

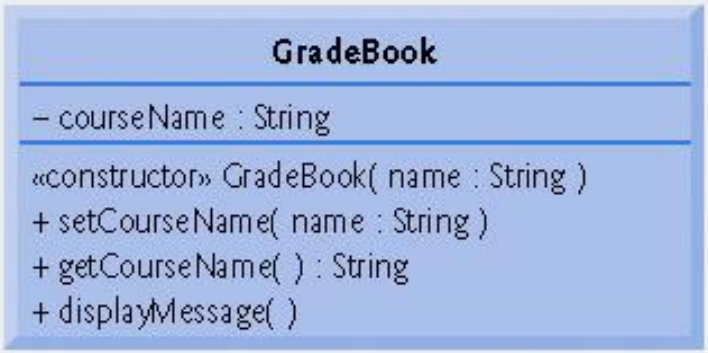

Figura 3.12 | Diagrama de classe UML indicando que a classe `GradeBook` tem um construtor que tem um parâmetro `name` do tipo UML `String`.

3.8 Números de ponto flutuante e tipo `double`

- **Números de ponto flutuante:**
  - `float`
  - `double`
    - **Armazenam números com maior magnitude e mais detalhes (isto é, mais dígitos à direita do ponto de fração decimal – também conhecido como *precisão* do número) do que as variáveis `float`.**

Precisão de número de ponto flutuante e requisitos de memória

- `float`
  - **Números de ponto flutuante de precisão simples.**
  - **Sete dígitos significativos.**
- `double`
  - **Números de ponto flutuante de dupla precisão.**
  - **Quinze dígitos significativos.**

Erro comum de programação 3.4

Utilizar números de ponto flutuante – de maneira tal que assume que eles são representados precisamente – pode levar a erros de lógica.

Resumo

Account.java

```
1  // Fig. 3.13: Account.java
2  // Classe Account com um construtor para
3  // inicializar a variável de instância balance.
4
5  public class Account
6  {
7     private double balance; // variável de instância que armazena o saldo
8
9     // construtor
10    public Account( double initialBalance )
11    {
12       // valida que initialBalance é maior que 0,0;
13       // se não, o saldo é inicializado como o valor padrão 0.0
14       if ( initialBalance > 0.0 )
15          balance = initialBalance;
16    } // fim do construtor Account
17
18    // credita (adiciona) uma quantia à conta
19    public void credit( double amount )
20    {
21       balance = balance + amount; // adiciona amount ao balance
22    } // end method credit
23
24    // retorna o saldo da conta
25    public double getBalance()
26    {
27       return balance; // fornece o valor de balance ao método chamador
28    } // fim do método getBalance
29
30 } // fim da classe Account
```

Variável `double balance`

Classe `AccountTest` para utilizar a classe `Account`

- **Especificador de formato `%f`:**
  - **Utilizado para gerar saída de números de ponto flutuante.**
  - **Posicione um decimal e um número entre o sinal de porcentagem e o `f` para impor uma precisão.**

Resumo

AccountTest.Java

(1 de 3)

```
1  // Fig. 3.14: AccountTest.java
2  // Cria e manipula um objeto Account.
3  import java.util.Scanner;
4
5  public class AccountTest
6  {
7     // método main inicia a execução do aplicativo Java
8     public static void main( String args[] )
9     {
10       Account account1 = new Account( 50.00 ); // cria o objeto Account
11       Account account2 = new Account( -7.53 ); // cria o objeto Account
12
13       // exibe saldo inicial de cada objeto
14       System.out.printf( "account1 balance: $%.2f\n",
15          account1.getBalance() );
16       System.out.printf( "account2 balance: $%.2f\n\n",
17          account2.getBalance() );
18
```

Resumo

AccountTest.java

```
19    // cria Scanner para obter entrada a partir da janela de comando
20    Scanner input = new Scanner( System.in );
21    double depositAmount; // quantia de depósito lida a partir do usuário
22
23    System.out.print( "Enter deposit amount for account1: " ); // prompt
24    depositAmount = input.nextDouble(); // obtém a entrada do usuário
25    System.out.printf( "\nadding %.2f to account1 balance\n\n"
26       depositAmount );
27    account1.credit( depositAmount ); // adiciona ao saldo de account1
28
29    // display balances
30    System.out.printf( "account1 balance: $%.2f\n",
31       account1.getBalance() );
32    System.out.printf( "account2 balance: $%.2f\n\n",
33       account2.getBalance() );
34
35    System.out.print( "Enter deposit amount for account2: " ); // prompt
36    depositAmount = input.nextDouble(); // obtém a entrada do usuário
37    System.out.printf( "\nadding %.2f to account2 balance\n\n",
38       depositAmount );
39    account2.credit( depositAmount ); // adiciona ao saldo de account2
40
```

Insere um valor `double`

Insere um valor `double`

Resumo

AccountTest.java

(3 de 3)

```
41       // exibe os saldos
42       System.out.printf( "account1 balance: $%.2f\n",
43          account1.getBalance() );
44       System.out.printf( "account2 balance: $%.2f\n",
45          account2.getBalance() );
46    } // fim de main
47
48 } // fim da classe AccountTest
```

Gera a saída de um valor `double`

```
account1 balance: $50.00
account2 balance: $0.00

Enter deposit amount for account1: 25.53

adding 25.53 to account1 balance

account1 balance: $75.53
account2 balance: $0.00

Enter deposit amount for account2: 123.45

adding 123.45 to account2 balance

account1 balance: $75.53
account2 balance: $123.45
```

Figura 3.15 | Diagrama da classe UML para indicar que a classe `Account` tem um atributo `private balance` do tipo UML `Double`, um construtor (com um parâmetro do tipo UML `Double`) e duas operações `public` – `credit` (com um parâmetro `amount` do tipo UML `Double`) e `getBalance` (retorna o tipo UML `Double`).

Local	Título — Exercício(s)
Seção 3.9	Utilizando caixas de diálogo — Entrada e saída básicas com caixas de diálogo
Seção 4.14	Criando desenhos simples — Exibindo e desenhando linhas na tela
Seção 5.10	Desenhando retângulos e ovais — Utilizando formas para representar dados
Seção 6.13	Cores e formas preenchidas — Desenhando um alvo e imagens gráficas aleatórias
Seção 7.13	Desenhando arcos — Desenhando espirais com arcos
Seção 8.18	Utilizando objetos com imagens gráficas — Armazenando formas como objetos
Seção 9.8	Exibindo texto e imagens utilizando rótulos — Fornecendo informações de status
Seção 10.8	Desenhando com polimorfismo — Identificando as semelhanças entre formas
Exercícios 11.18	Expandindo a interface — Utilizando componentes GUI e tratamento de evento
Exercícios 12.12	Adicionando Java 2D — Utilizando a API Java 2D para aprimorar desenhos

Figura 3.16 | Resumo da GUI e estudo de caso de imagens gráficas em cada capítulo.

Exibindo texto em uma caixa de diálogo

- **Janelas e caixas de diálogo:**
  - **Muitos aplicativos Java utilizam janelas e caixas de diálogo para exibir a saída.**
  - `JOptionPane` **fornece caixas de diálogo pré-empacotadas chamadas *diálogos de mensagem***

Resumo

Dialog1.java

```
1  // Fig. 3.17: Dialog1.java
2  // Imprimindo múltiplas linhas na caixa de diálogo.
3  import javax.swing.JOptionPane; // importa a classe JOptionPane
4
5  public class Dialog1
6  {
7     public static void main( String args[] )
8     {
9        // exibe um diálogo com a mensagem
10       JOptionPane.showMessageDialog( null, "Welcome\nto\nJava" );
11    } // fim de main
12 } // fim da classe Dialog1
```

Importa a classe `JOptionPane`

Mostra um diálogo de mensagem com texto

Exibindo texto em uma caixa de diálogo

- **Pacote `javax.swing`:**
  - **Contém classes para ajudar a criar interfaces gráficas com o usuário (GUIs).**
  - **Contém a classe `JOptionPane`.**
    - **Declara o método `static` `showMessageDialog` para exibir um diálogo de mensagem.**

Inserindo texto em uma caixa de diálogo

- **Diálogo de entrada:**
  - **Permite que o usuário insira informações.**
  - **Criado com o método** `showInputDialog` **na classe** `JOptionPane`.

Resumo

NameDialog.Java

```
1  // Fig. 3.18: NameDialog.java
2  // Entrada básica com uma caixa de diálogo.
3  import javax.swing.JOptionPane;
4
5  public class NameDialog
6  {
7     public static void main( String args[] )
8     {
9        // pede para o usuário inserir seu nome
10       String name =
11          JOptionPane.showInputDialog( "What is your name?" );
12
13       // cria a mensagem
14       String message =
15          String.format( "Welcome, %s, to Java Programming!", name );
16
17       // exibe a mensagem para cumprimentar o usuário pelo nome
18       JOptionPane.showMessageDialog( null, message );
19    } // fim de main
20 } // fim da classe NameDialog
```

Mostra o diálogo de entrada

Formata uma `String` para gerar saída ao usuário

3.10 (Opcional) Estudo de caso de engenharia de software: Identificando classes em um documento de requisitos

- **Começa a projetar o sistema ATM:**
  - **Analisa os substantivos simples e os substantivos compostos.**
  - **Introduz diagramas de classe UML.**

Identificando as classes em um sistema

- **Principais substantivos simples e substantivos compostos em um documento de requisitos:**
  - **Alguns são atributos de outras classes.**
  - **Alguns não correspondem a partes do sistema.**
  - **Alguns são classes.**
    - **Classes a ser representadas por diagramas de classe UML.**

Substantivos simples e substantivos compostos no documento de requisitos		
banco	**dinheiro / fundos**	**número de conta**
ATM	**tela**	**PIN**
usuário	**teclado**	**banco de dados do banco**
cliente	**dispensador de cédulas (*cash dispenser*)**	**pesquisa de saldo**
transação	**cédula de US$ 20 / dinheiro**	**retirada/saque**
conta	**entrada de depósito**	**depósito**
saldo	**envelope de depósito**	

Figura 3.19 | Substantivos simples e substantivos compostos no documento de requisitos.

Modelando classes

- **Diagramas de classe UML:**
  - **Compartimento superior contém o nome da classe centralizada horizontalmente em negrito.**
  - **Compartimento intermediário contém atributos da classe ou variáveis de instância.**
  - **Compartimento inferior contém as operações ou métodos da classe.**

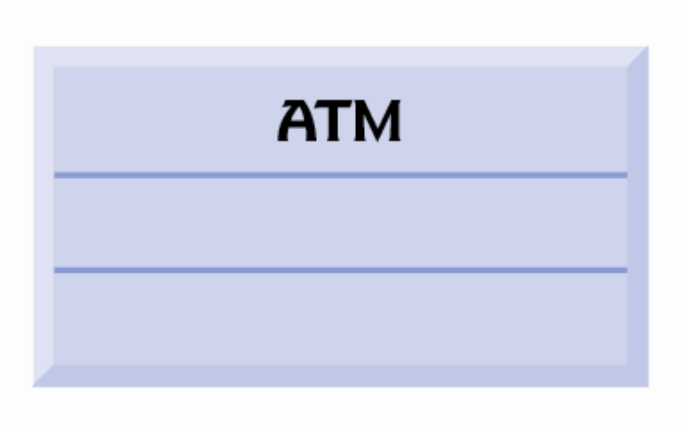

Figura 3.20 | Representando uma classe na UML utilizando um diagrama de classe.

Modelando classes (*Continuação*)

- **A UML:**
  - **Permite a supressão de atributos e operações de classe para criar diagramas mais legíveis, quando apropriado — chamados de *diagramas elididos*.**
  - **A linha sólida que conecta duas classes representa uma *associação*.**
    - **Números próximos do final de cada linha são valores de *multiplicidade*.**

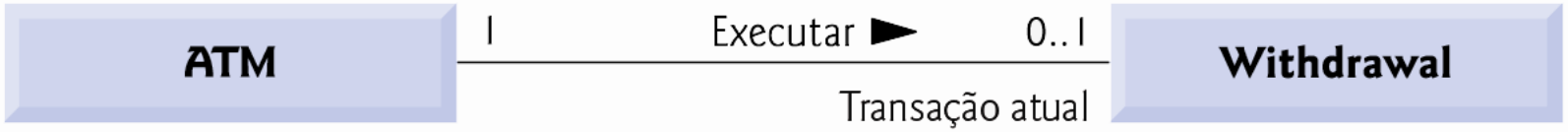

Figura 3.21 | Diagrama de classes que mostra uma associação entre classes.

Símbolo	Significado
0	Nenhum
1	Um
m	Um valor de inteiro
0..1	Zero ou um
m, *n*	*m* ou *n*
m..*n*	Pelo menos *m*, mas não mais do que *n*
*	Qualquer inteiro não-negativo (zero ou mais)
0..*	Zero ou mais (idêntico a *)
1..*	Um ou mais

Figura 3.22 | Tipos de multiplicidade.

Modelando classes (*Continuação*)

- **Diagramas de classe UML:**
  - **Losangos sólidos anexados a linhas de associação indicam um relacionamento de composição.**
  - **Losangos ocos indicam agregação — uma forma mais fraca de composição.**

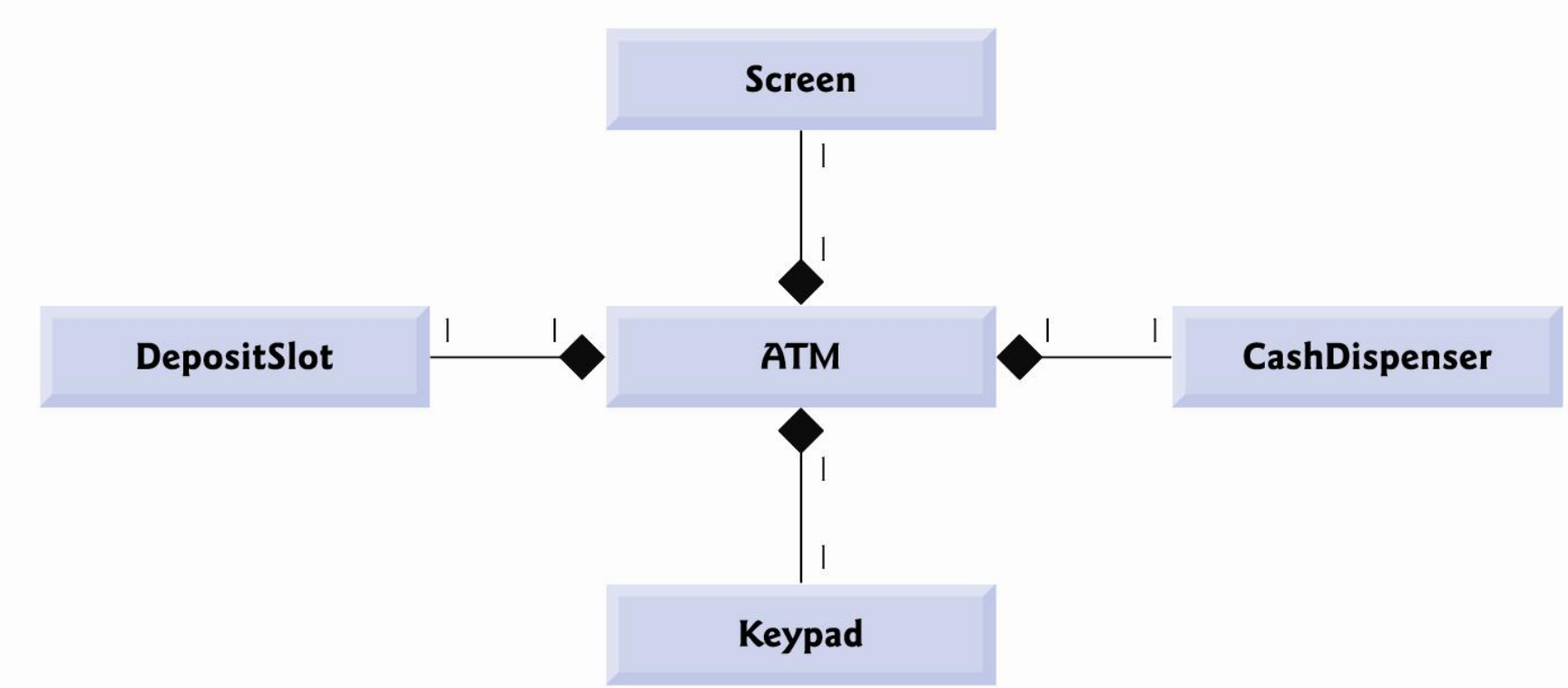

Figura 3.23 | Diagrama de classes mostrando os relacionamentos de composição.

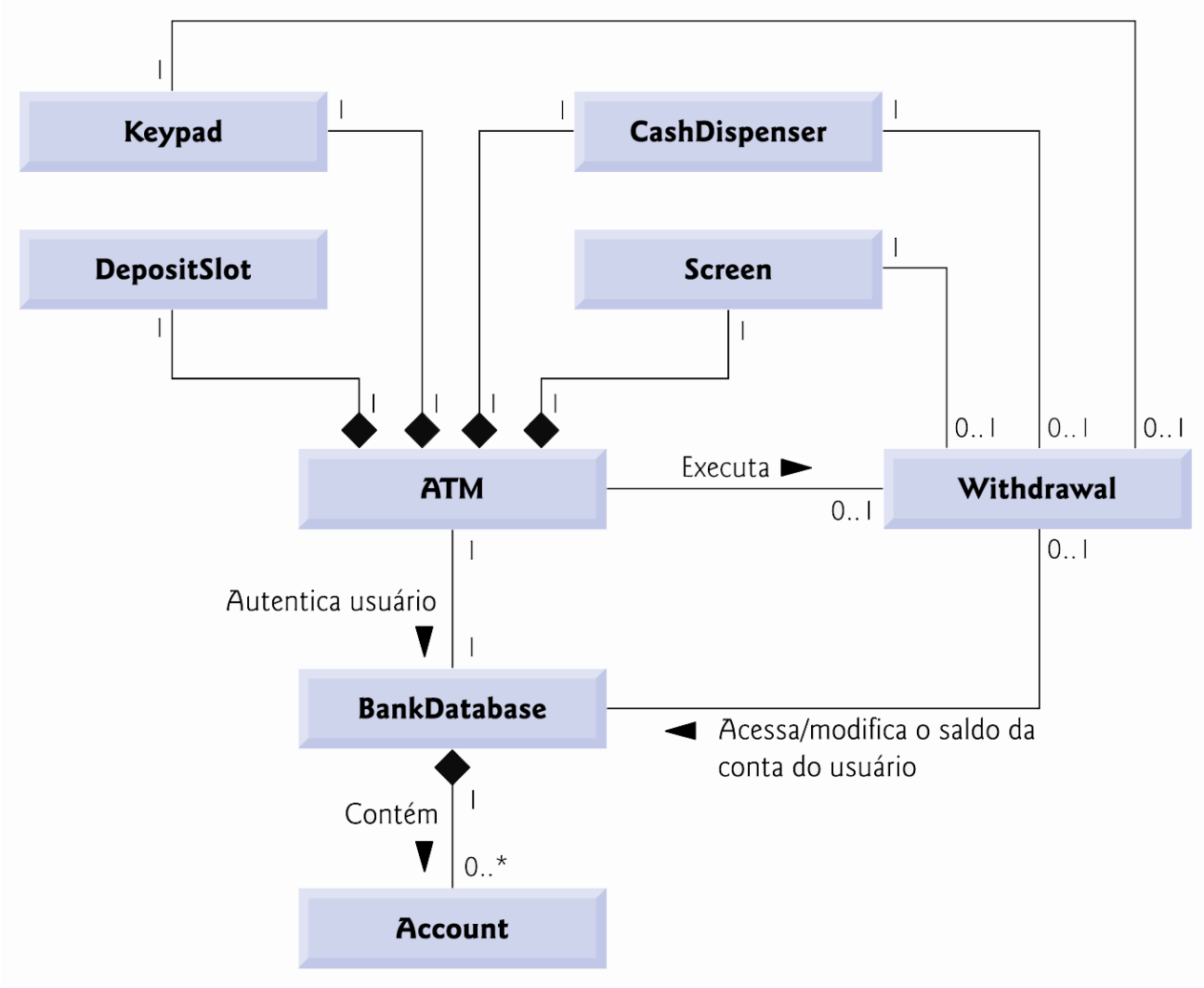

Figura 3.24 | Diagrama de classes para o modelo do sistema ATM.

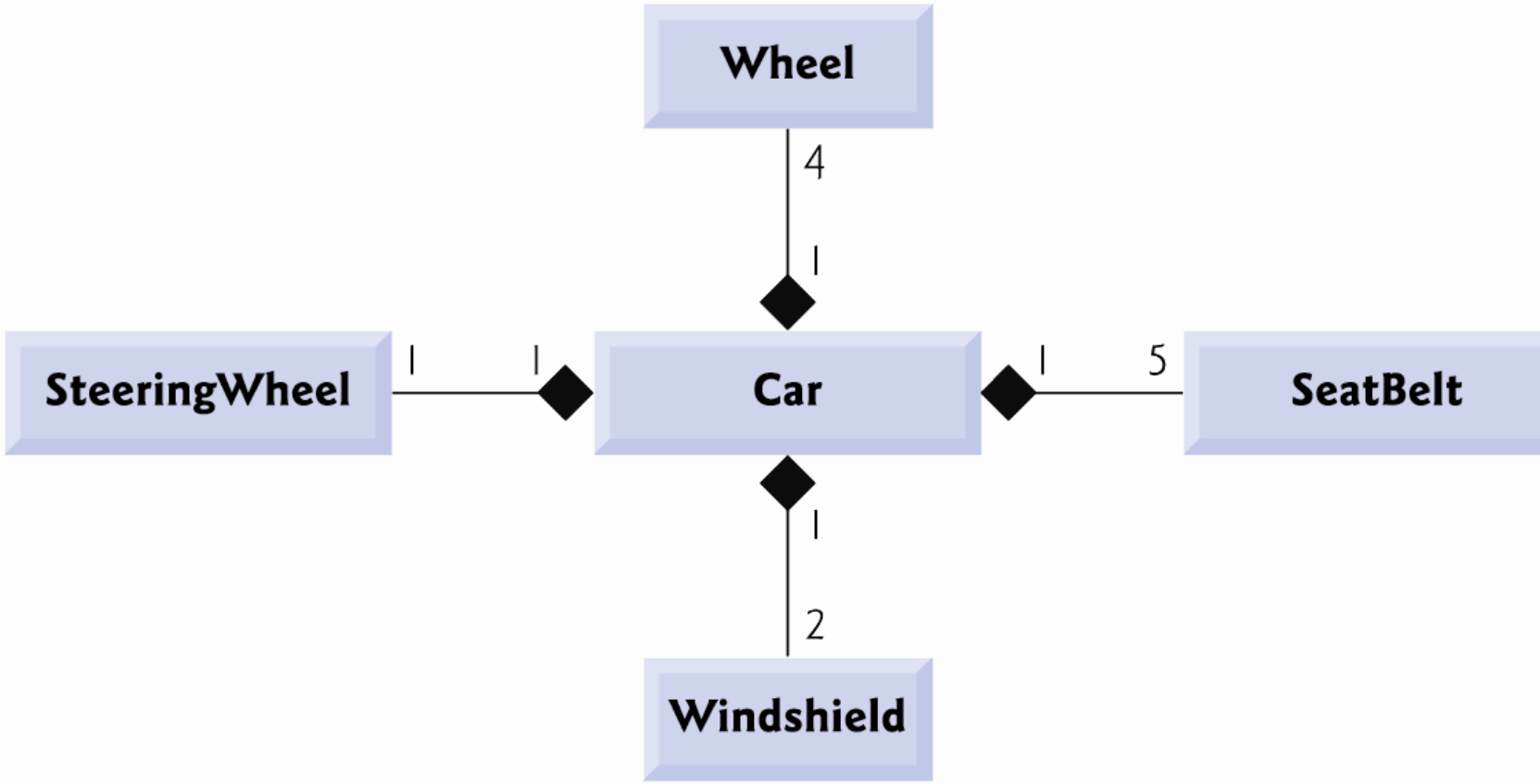

Figura 3.25 | Diagrama de classes mostrando relacionamentos de composição de uma classe `Car`.

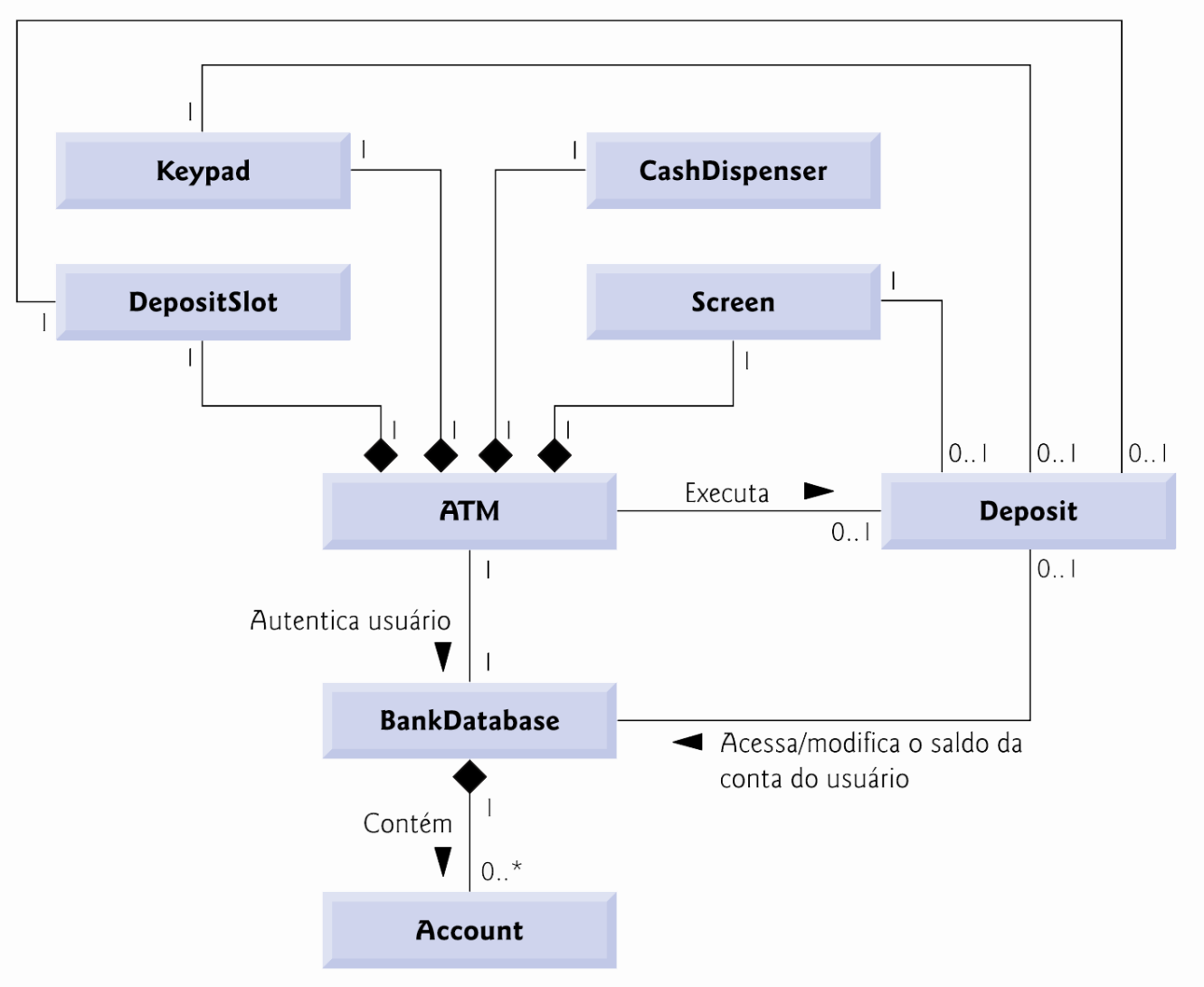

Figura 3.26 | Diagrama de classes para o modelo de sistema ATM incluindo a classe `Deposit`.