Java: como programar
Programação Orientada à Objetos

- Douglas Baptista de Godoy

/in/douglasbgodoy

github.com/douglasbgodoy

Fatec
Faculdade de Tecnologia

Etec
Escola Técnica Estadual

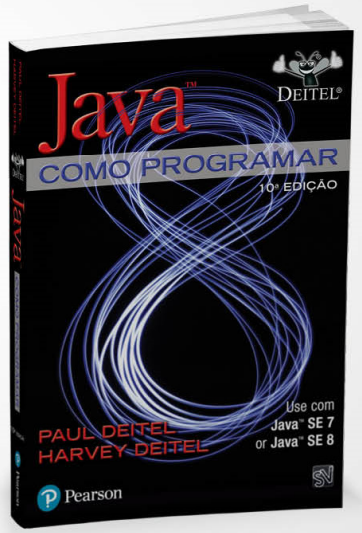
INSTITUTO FEDERAL
DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
São Paulo

Informação

Obs: Esta aula é baseada nos livros textos, e as transparências são baseadas nas transparências providenciadas pelos autores.

DEITEL, P. J.; DEITEL, H. M. **Java:** como programar. 10. ed. São Paulo, SP: Pearson, 2017. *E-book*. Disponível em: https://plataforma.bvirtual.com.br. Acesso em: 27 fev. 2024.

Capítulo 3:
Introdução a classes, objetos, métodos e strings



Classes, objetos, métodos e variáveis de instância

- **Analogia simples para ajudar a entender classes e seu conteúdo.**

  Suponha que você queira guiar um carro e fazê-lo andar mais rápido pisando no pedal acelerador.

  Antes de poder dirigir um carro, alguém tem de projetá-lo.

  Em geral, um carro inicia com os desenhos de engenharia, semelhantes às plantas utilizadas para projetar uma casa.

  Estes incluem o projeto de um pedal acelerador para aumentar a velocidade do carro.

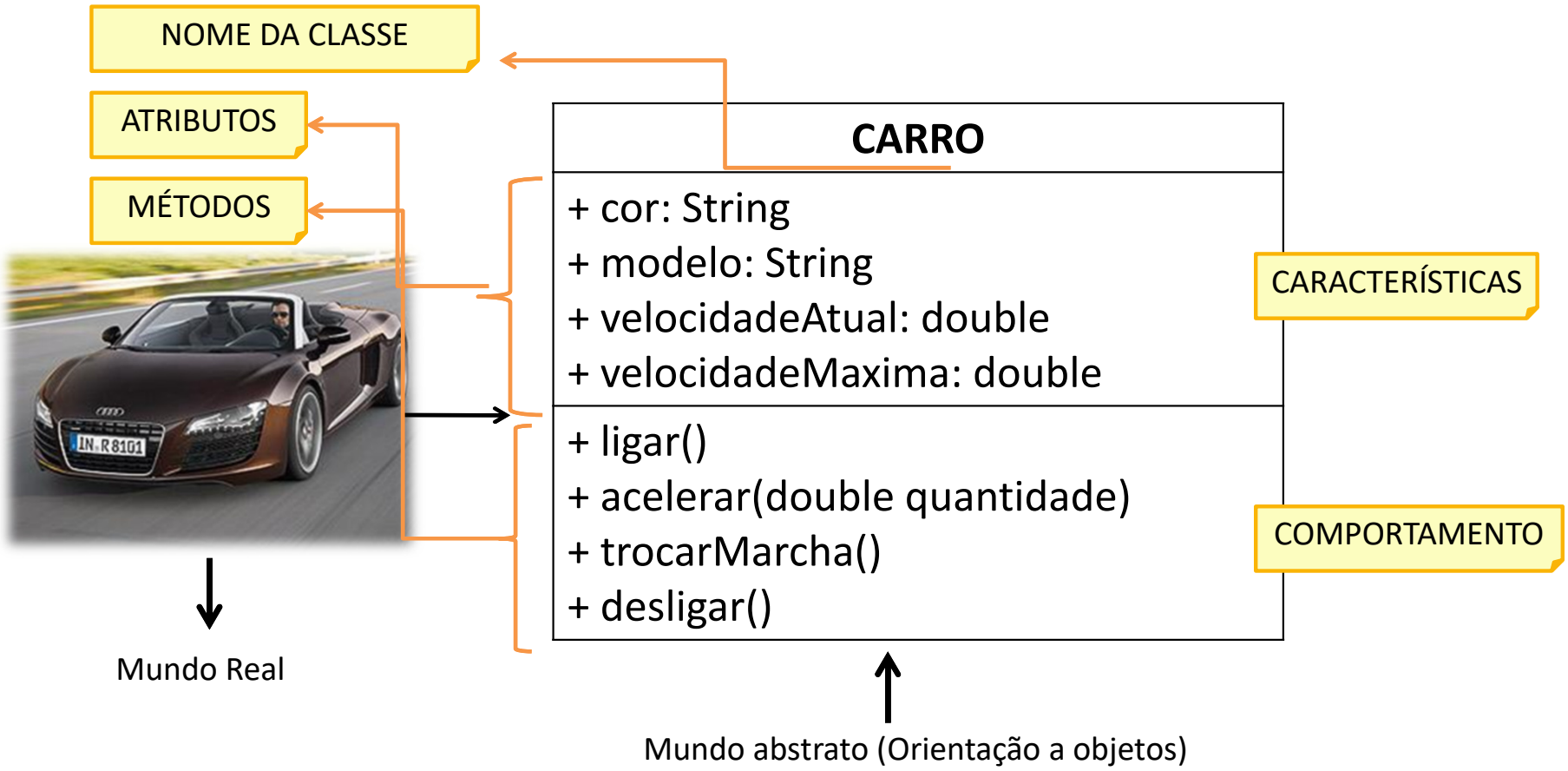
- **Analogia simples para ajudar a entender classes e seu conteúdo.**

O pedal "oculta" do motorista os complexos mecanismos que realmente fazem o carro ir mais rápido, assim como o pedal de freio "oculta" os mecanismos que diminuem a velocidade do carro e a direção "oculta" os mecanismos que mudam a direção do carro.

Isso permite que as pessoas com pouco ou nenhum conhecimento de como os motores funcionam dirijam um carro facilmente.

Antes de poder guiar um carro, ele deve ser construído a partir dos desenhos de engenharia que o descrevem.

Um carro pronto tem um pedal de acelerador real para fazer o carro andar mais rápido, mas até isso não é suficiente — o carro não acelerará por conta própria, então o motorista deve pressionar o pedal do acelerador.


NOME DA CLASSE
ATRIBUTOS
MÉTODOS
CARRO
+ cor: String
+ modelo: String
+ velocidadeAtual: double
+ velocidadeMaxima: double
+ ligar()
+ acelerar(double quantidade)
+ trocarMarcha()
+ desligar()
CARACTERÍSTICAS
COMPORTAMENTO
Mundo Real
Mundo abstrato (Orientação a objetos)

- **Para realizar uma tarefa em um programa é necessário um método.**

  O método descreve os mecanismos que realmente realizam suas tarefas.

  A função oculta de seu usuário as tarefas complexas que ele realiza, assim como o pedal acelerador de um carro oculta do motorista os complexos mecanismos que fazem o carro andar mais rápido.

- Em Java, uma classe abriga um método, assim como os desenhos de engenharia do carro abrigam o projeto de um pedal acelerador.

- Em uma classe, você fornece um ou mais métodos que são projetados para realizar as tarefas da classe.

- **Analogia simples para ajudar a entender Objetos.**
- Você deve criar um objeto de uma classe antes de um programa realizar as tarefas que a classe descreve como fazer.

  Essa é uma razão por que o Java é conhecido como uma linguagem de programação orientada a objetos.
- Ao dirigir um carro, o ato de pressionar o acelerador envia uma mensagem para o carro realizar uma tarefa — fazer o carro andar mais rápido.
- Você **envia mensagens** para um objeto — cada mensagem é implementada como uma **chamada de método** que instrui um método do objeto a realizar sua tarefa.

- **Analogia simples para ajudar a entender Atributos.**

- Um carro tem muitos atributos

  Cor, o número de portas, a capacidade do tanque, a velocidade atual e a quilometragem.
- Atributos são representados como parte do projeto de um carro nos diagramas de engenharia.
- Cada carro mantém seus próprios atributos.

  Cada carro sabe a quantidade de gasolina que há no seu tanque, mas não sabe quanto há no tanque de outros carros.

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

- 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.

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

- Os nomes de classe, método e variável são identificadores. Os nomes de classe começam com letra maiúscula, e os de método e variável, com uma letra minúscula.

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

```
1  // Figura 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 } // fim da classe GradeBook
```

Executa a tarefa de exibir uma mensagem na tela; o método `displayMessage` deve ser chamado para realizar essa tarefa

Figura 3.1 | Declaração de classe com um método.

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

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

Cria o objeto `GradeBook` e o atribui à variável `myGradeBook`

Invoca o método `displayMessage` no objeto GradeBook que foi atribuído à variável `myGradebook`

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

Figura 3.2 | Criando um objeto `GradeBook` e chamando seu método `displayMessage`.

Diagrama de classe de UML.

- Figura 3.3: Diagrama de classe de UML para a classe `GradeBook`.
- Cada classe é modelada num diagrama de classe como um retângulo com três compartimentos.
  Parte superior: contém o nome de classe centraliazdo horizontalmente em tipo negrito.
  Meio: contém os atributos da classe, que correspondem a variáveis de exemplo (Seção 3.5).
  Parte inferior: contém as operações da classe, que correspondem a métodos.
- Operações são modeladas listando o nome da operação precedido por um modificador de acesso (nesse caso +) e seguido por um conjunto de parêntesis.
- O sinal de adição (+) corresponde à palavra-chave `public`.

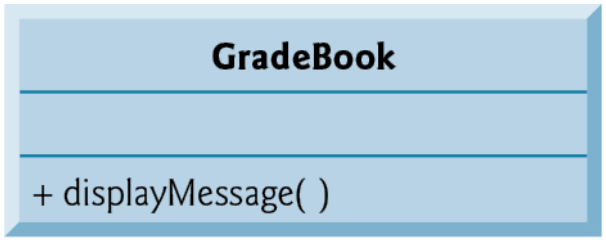


Figura 3.3 | Diagrama de classe UML indicando que a classe `GradeBook` tem uma operação `public displayMessage`.

3.4 Declarando um método com um parâmetro

```
1  // Figura 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 de 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 } // fim da classe GradeBook
```

O parâmetro `courseName` fornece as informações adicionais que o método requer para executar sua tarefa

O valor do parâmetro `courseName` é exibido como parte da saída

Figura 3.4 | Declaração de classe com um método que tem um parâmetro.

3.4 Declarando um método com um parâmetro

```
1  // Figura 3.5: GradeBookTest.java
2  // Cria objeto GradeBook e passa uma String para
3  // seu método displayMessage.
4  import java.util.Scanner; // programa utiliza Scanner
5
6  public class GradeBookTest
7  {
8     // método main inicia a execução de programa
9     public static void main( String[] args )
10    {
11       // cria Scanner para obter entrada a partir da janela de comando
12       Scanner input = new Scanner( System.in );
13
14       // cria um objeto GradeBook e o atribui a myGradeBook
15       GradeBook myGradeBook = new GradeBook();
16
17       // prompt para entrada do nome do curso
18       System.out.println( "Please enter the course name:" );
19       String nameOfCourse = input.nextLine(); // lê uma linha de texto
20       System.out.println(); // gera saída de uma linha em branco
21
```

Lê uma linha de texto

Figura 3.5 | Criando um objeto `GradeBook` e passando uma `String` ao seu método `displayMessage`. (Parte 1 de 2.)

3.4 Declarando um método com um parâmetro

```
22          // chama método displayMessage de myGradeBook
23          // e passa nameOfCourse como um argumento
24          myGradeBook.displayMessage( nameOfCourse );
25       } // fim de main
26    } // fim da classe GradeBookTest
```

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

Welcome to the GradeBook for
CS101 Introduction to Java Programming!
```

Figura 3.5 | Criando um objeto `GradeBook` e passando uma `String` ao seu método `displayMessage`. (Parte 2 de 2.)

Diagrama de classe de UML.

3.4 Declarando um método com um parâmetro

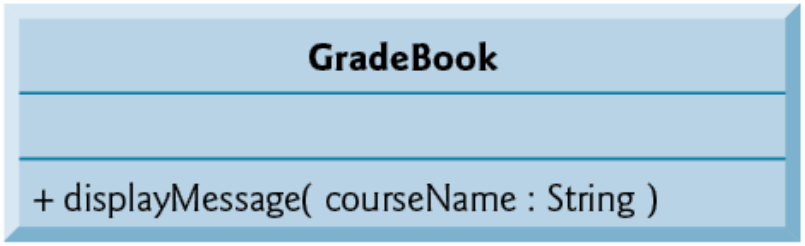


Figura 3.6 | Diagrama de classe UML indicando que a classe `GradeBook` tem uma operação `displayMessage` com um parâmetro `courseName` de tipo UML `String`.

Declarando método set e método get

Atributo (variável de instância), um método *set* e um método *get*

- A maioria das declarações de variável de instância (Atributo) 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.

método *set* e método *get*

- 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.

método *set* e método *get*

- 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.

método *set* e método *get*

- 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.

método *set* e método *get*

- 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.

método *set* e método *get*

- 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`. Essa convenção de nomenclatura não é obrigatória, mas é recomendada.

Declarando método set e método get

```
1  // Figura 3.7: GradeBook.java
2  // classe GradeBook que contém uma variável de instância
3  // courseName e métodos para configurar e obter seu valor.
4
5  public class GradeBook
6  {
7     private String courseName; // nome do curso para esse 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
```

Cada objeto GradeBook mantém sua própria cópia da variável de instância courseName

Método que permite ao cliente codificar para alterar o courseName

Método que permite ao cliente codificar para obter o courseName

Figura 3.7 | A classe GradeBook que contém uma variável de instância courseName e métodos para configurar e obter seu valor. (Parte 2 de 2.)

```
21      // exibe uma mensagem de boas-vindas para o usuário GradeBook
22      public void displayMessage()
23      {
24         // chama getCourseName para obter o nome do
25         // o curso que essa GradeBook representa
26         System.out.printf( "Welcome to the GradeBook for\n%s!\n",
27            getCourseName() );
28      } // fim do método displayMessage
29   } // fim da classe GradeBook
```

Nenhum parâmetro requerido; todos os métodos nesta classe já sabem sobre a variável de instância `courseName` e os outros métodos da classe

Boa prática para acessar suas variáveis de instância via métodos *set* ou *get*

Figura 3.7 | A classe `GradeBook` que contém uma variável de instância `courseName` e métodos para configurar e obter seu valor. (Parte 2 de 2.)

Cria e usa um objeto

- A expressão de criação de instância de classe começa com a palavra-chave `new` e estabelece um novo objeto.

```
1  // Figura 3.8: GradeBookTest.java
2  // Criando e manipulando 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
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 ); // configura o nome do curso
```

Obtém o valor da variável de instância `courseName` do objeto `myGradeBook`

Configura o valor da variável de instância `courseName`

Figura 3.8 | Criando e manipulando um objeto `GradeBook`. (Parte 1 de 2.)

```
24          System.out.println(); // gera saída de uma linha em branco
25          // exibe mensagem de boas-vindas depois de especificar o
26          // nome do curso
27          myGradeBook.displayMessage();
28       } // fim de main
29    } // fim da classe GradeBookTest
```

Exibe a mensagem do GradeBook, incluindo o valor da variável de instância `courseName`

```
Initial course name is: null

Please enter the course name:
CS101 Introduction to Java Programming

Welcome to the GradeBook for
CS101 Introduction to Java Programming!
```

Figura 3.8 | Criando e manipulando um objeto `GradeBook`. (Parte 2 de 2.)

Diagrama de classe de UML.

Declarando método set e método get

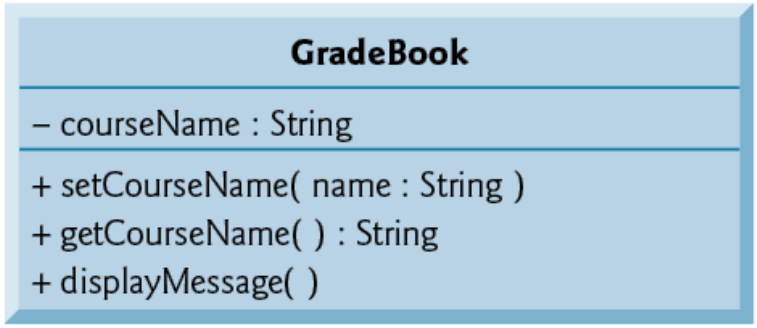


Figura 3.9 | O diagrama de classes UML que indica que a classe `GradeBook` tem um atributo `courseName` privado do tipo UML `String` e três operações públicas — `setCourseName` (com um parâmetro `name` do tipo UML `String`), `getCourseName` (que retorna o tipo UML `String`) e `displayMessage`.

Inicializando objetos com construtores

inicialização de objetos com construtores

• 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.

3.8 Inicializando objetos com construtores

```
1  // Figura 3.10: GradeBook.java
2  // Classe GradeBook com um construtor para inicializar o
3  // nome de um curso.
4  public class GradeBook
5  {
6     private String courseName; // nome do curso para esse GradeBook
7
8     // o construtor inicializa courseName com o argumento String
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
```

Construtor que inicializa `courseName` com um argumento

Figura 3.10 | A classe `GradeBook` com um construtor para inicializar o nome do curso. (Parte 1 de 2.)

```
20       // método para recuperar o nome do curso
21       public String getCourseName()
22       {
23          return courseName;
24       } // fim do método getCourseName
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 GradeBook for\n%s!\n",
32             getCourseName() );
33       } // fim do método displayMessage
34    } // fim da classe GradeBook
```

Figura 3.10 | A classe GradeBook com um construtor para inicializar o nome do curso. (Parte 2 de 2.)

Inicializando objetos com construtores

- 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.

```
1  // Figura 3.11: GradeBookTest.java
2  // construtor GradeBook utilizado para especificar o nome
3  // do curso na hora em que cada objeto GradeBook é criado.
4
5  public class GradeBookTest
6  {
7     // método main inicia a execução de programa
8     public static void main( String[] args )
9     {
10       // cria objeto GradeBook
11       GradeBook gradeBook1 = new GradeBook(
12          "CS101 Introduction to Java Programming" );
13       GradeBook gradeBook2 = new GradeBook(
14          "CS102 Data Structures in Java" );
15
16       // exibe valor inicial de courseName para cada 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    } // fim de main
22 } // fim da classe GradeBookTest
```

A expressão de criação de instância de classe inicializa GradeBook e retorna uma referência que é atribuída à variável gradeBook1

A expressão de criação de instância de classe inicializa GradeBook e retorna uma referência que é atribuída à variável gradeBook2

Figura 3.11 | O construtor de GradeBook usado para especificar o nome do curso no momento em que cada objeto GradeBook é criado. (Parte 1 de 2.)

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

Figura 3.11 | O construtor de `GradeBook` usado para especificar o nome do curso no momento em que cada objeto `GradeBook` é criado. (Parte 2 de 2.)

Diagrama de classe de UML.

Inicializando objetos com construtores

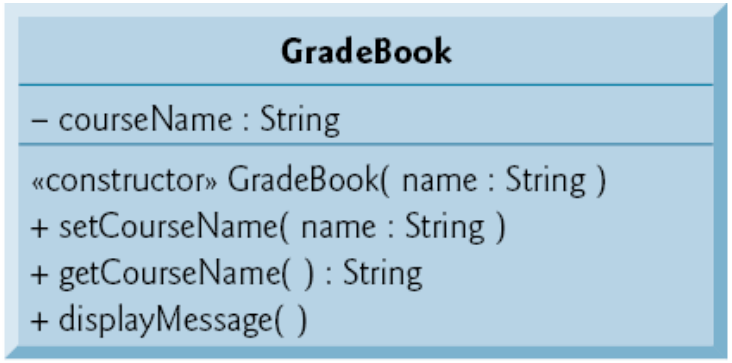


Figura 3.12 | Diagrama de classe UML que indica que a classe `GradeBook` tem um construtor com um parâmetro `name` de tipo UML `String`.

Declarando um método com um parâmetro

- 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.

A classe `Account`

```
1  // Figura 3.13: Account.java
2  // classe Account com um construtor para validar e
3  // inicializa a variável de instância balance do tipo double.
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 quantia ao saldo
22    } // fim do método credit
```

Número de ponto flutuante para o saldo da conta

Parâmetro utilizado para inicializar a variável de instância `balance`

Valida o valor do parâmetro para assegurar que ele é maior que 0

Inicializa `gradeCounter` como 1; indica que a primeira nota está prestes a ser inserida

Figura 3.13 | A classe `account` com um construtor para validar e inicializar a variável de instância `balance` do tipo `double`. (Parte 1 de 2.)

```
23
24      // retorna o saldo da conta
25      public double getBalance()
26      {
27         return balance; // fornece o valor de saldo ao método chamador
28      } // fim do método getBalance
29   } // fim da classe Account
```

Retorna o valor da variável de instância `balance` como `double`

Figura 3.13 | A classe `account` com um construtor para validar e inicializar a variável de instância `balance` do tipo `double`. (Parte 2 de 2.)

```
1  // Figura 3.14: AccountTest.Java
2  // Entrada e saída de números de ponto flutuante com objetos 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 o 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
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
```

Valores de ponto flutuante com dois dígitos de precisão

Figura 3.14 | Entrada e saída de números de ponto flutuante com objetos `Account`. (Parte 1 de 3.)

```
22
23          System.out.print( "Enter deposit amount for account1: " ); // prompt
24          depositAmount = input.nextDouble(); // entrada do usuário
25          System.out.printf( "\nadding %.2f to account1 balance\n\n",
26             depositAmount );
27          account1.credit( depositAmount ); // adiciona o saldo de account1
28
29          // exibe os saldos
30          System.out.printf( "account1 balance: $%.2f \n",
31             account1.getBalance() );
32          System.out.printf( "account2 balance: $%.2f \n\n",
33             account2.getBalance() );
34          // prompt
35          System.out.print( "Enter deposit amount for account2: " );
36          depositAmount = input.nextDouble(); // entrada do usuário
37          System.out.printf( "\nadding %.2f to account2 balance\n\n",
38             depositAmount );
39          account2.credit( depositAmount ); // adiciona ao saldo
40                                            // de account2
41          // exibe os saldos
42          System.out.printf( "account1 balance: $%.2f \n",
43             account1.getBalance() );
```

Retorna um valor `double` digitado pelo usuário

Figura 3.14 | Entrada e saída de números de ponto flutuante com objetos `Account`. (Parte 2 de 3.)

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

```
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 3.14 | Entrada e saída de números de ponto flutuante com objetos `Account`. (Parte 3 de 3.)

Diagrama de classe de UML.

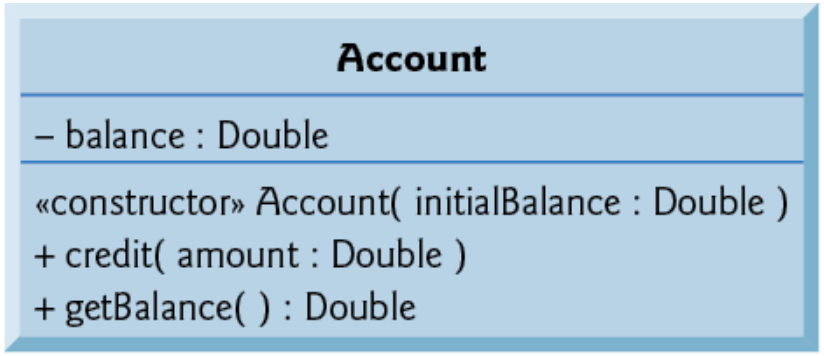


Figura 3.15 | O diagrama de classes UML indicando que a classe `Account` tem um atributo `private balance` do tipo UML `Double`, um construtor (com um parâmetro do tipo UML `Double`) e duas operações `public` — `credit` (com um parâmetro `amount` do tipo UML `Double`) e `getBalance` (retorna o tipo UML `Double`).

Estudo de caso de GUI e imagens gráficas: utilizando caixas de diálogo

- Figura 3.16: Resumo do “Estudo de caso GUI e imagens gráficas” em cada capítulo.

Exibindo texto em uma caixa de diálogo

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

Importa a classe `JOptionPane` para ser usada neste programa

Exibe o diálogo de mensagem no centro da tela

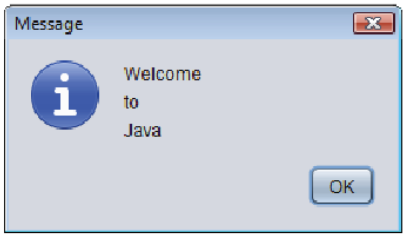


Figura 3.17 | Utilizando `JOptionPane` para exibir múltiplas linhas em uma caixa de diálogo.

Inserindo texto em uma caixa de diálogo

```
1  // Figura 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 } // termina NameDialog
```

Exibe um diálogo de entrada para obter dados do usuário

Cria uma String formatada contendo o nome inserido pelo usuário na caixa de diálogo

Figura 3.18 | Obtendo a entrada de usuário a partir de um diálogo. (Parte 1 de 2)

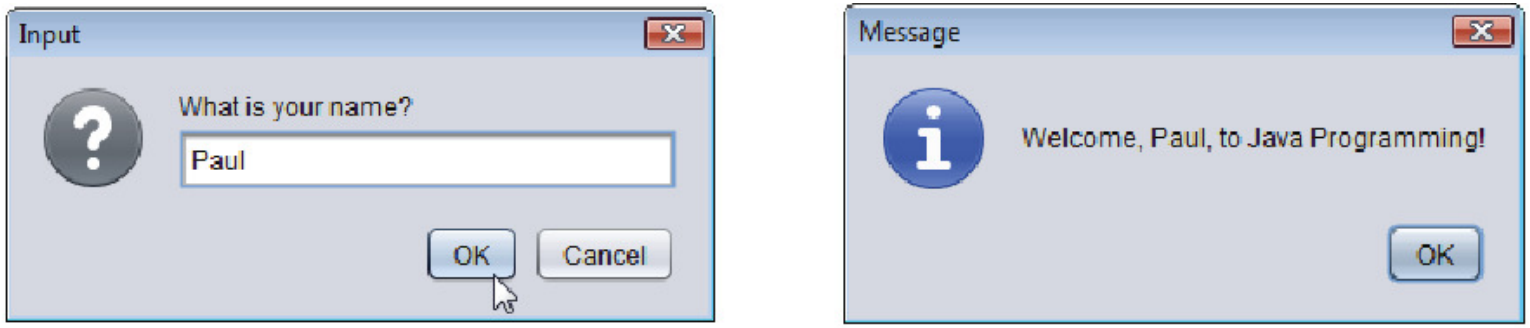


Figura 3.18 | Obtendo a entrada de usuário a partir de um diálogo. (Parte 2 de 2.)

Referências Bibliográficas

- DEITEL, P. J.; DEITEL, H. M. **Java:** como programar. 10. ed. São Paulo, SP: Pearson, 2017.