

Java: como programar Programação Orientada à Objetos

- Douglas Baptista de Godoy

 [/in/douglasbgodoy](https://www.linkedin.com/in/douglasbgodoy)

 github.com/douglasbgodoy

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 8:

Classes e objetos: um exame mais profundo



8.8 Composição

- Uma classe pode ter referências a objetos de outras classes como membros.
- Isso é chamado **composição** e, às vezes, é referido como um **relacionamento tem um**.

8.8 Composição

```
1 // Figura 8.7: Date.java
2 // Declaração da classe Date.
3
4 public class Date
5 {
6     private int month; // 1-12
7     private int day; // 1-31 conforme o mês
8     private int year; // qualquer ano
9
10    private static final int[] daysPerMonth =
11        { 0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
12
13    // construtor: confirma o valor adequado para o mês e dia dado o ano
14    public Date(int month, int day, int year)
15    {
```

continua

8.8 Composição

```
16     // verifica se mês está no intervalo
17     if (month <= 0 || month > 12)
18         throw new IllegalArgumentException(
19             "month (" + month + ") must be 1-12");
20
21     // verifica se day está no intervalo para month
22     if (day <= 0 ||
23         (day > daysPerMonth[month] && !(month == 2 && day == 29)))
24         throw new IllegalArgumentException("day (" + day +
25             ") out-of-range for the specified month and year");
26
27     // verifique no ano bissexto se o mês é 2 e o dia é 29
28     if (month == 2 && day == 29 && !(year % 400 == 0 ||
29         (year % 4 == 0 && year % 100 != 0)))
30         throw new IllegalArgumentException("day (" + day +
31             ") out-of-range for the specified month and year");
32
33     this.month = month;
34     this.day = day;
35     this.year = year;
36
```

continuação

8.8 Composição

```
37     System.out.printf(  
38         "Date object constructor for date %s%n", this);  
39     }  
40  
41     // retorna uma String no formato mês/dia/ano  
42     public String toString()  
43     {  
44         return String.format("%d/%d/%d", month, day, year);  
45     }  
46 } // fim da classe Date
```

8.8 Composição

```
1 // Figura 8.8: Employee.java
2 // Classe Employee com referência a outros objetos.
3
4 public class Employee
5 {
6     private String firstName;
7     private String lastName;
8     private Date birthDate;
9     private Date hireDate;
10
11    // construtor para inicializar nome, data de nascimento e data de contratação
12    public Employee(String firstName, String lastName, Date birthDate,
13                    Date hireDate)
14    {
15        this.firstName = firstName;
16        this.lastName = lastName;
17        this.birthDate = birthDate;
18        this.hireDate = hireDate;
```

continua

8.8 Composição

```
19     }
20
21     // converte Employee em formato de String
22     public String toString()
23     {
24         return String.format("%s, %s Hired: %s Birthday: %s",
25             lastName, firstName, hireDate, birthDate);
26     }
27 } // fim da classe Employee
```

continuação

8.8 Composição

```
1 // Figura 8.9: EmployeeTest.java
2 // Demonstração de composição.
3
4 public class EmployeeTest
5 {
6     public static void main(String[] args)
7     {
8         Date birth = new Date(7, 24, 1949);
9         Date hire = new Date(3, 12, 1988);
10        Employee employee = new Employee("Bob", "Blue", birth, hire);
11
12        System.out.println(employee);
13    }
14 } // fim da classe EmployeeTest
```

```
Date object constructor for date 7/24/1949
Date object constructor for date 3/12/1988
Blue, Bob Hired: 3/12/1988 Birthday: 7/24/1949
```

8.9 Tipos enum

- Todos os tipos enum são tipos *por referência*.
- Um tipo enum é declarado com uma declaração enum, que é uma lista separada por vírgulas de constantes enum. A declaração pode incluir opcionalmente outros componentes das classes tradicionais, como construtores, campos e métodos.
- Constantes enum são implicitamente final, porque declaram constantes que não devem ser modificadas.
- Constantes enum são implicitamente static.

8.9 Tipos enum

- Qualquer tentativa de criar um objeto de um tipo enum com um operador new resulta em um erro de compilação.
- Constantes enum podem ser utilizadas em qualquer lugar em que constantes podem ser usadas, como nos rótulos case das instruções switch e para controlar instruções for aprimoradas.
- Cada constante enum em uma declaração enum é opcionalmente seguida por argumentos que são passados para o construtor enum.
- Para cada enum, o compilador gera um método static chamado values que retorna um array das constantes do enum na ordem em que elas foram declaradas.

8.9 Tipos enum

- O método `EnumSet static range` recebe as primeiras e últimas constantes `enum` em um intervalo e retorna um `EnumSet` que contém todas as constantes entre essas duas constantes, inclusive.

8.9 Tipos enum

```
1 // Figura 8.10: Book.java
2 // Declarando um tipo enum com um construtor e campos de instância explícitos
3 // e métodos de acesso para esses campos
4
5 public enum Book
6 {
7     // declara constantes do tipo enum
8     JHTP("Java How to Program", "2015"),
9     CHTP("C How to Program", "2013"),
10    IW3HTP("Internet & World Wide Web How to Program", "2012"),
11    CPPHTP("C++ How to Program", "2014"),
12    VBHTP("Visual Basic How to Program", "2014"),
13    CSHARPHTP("Visual C# How to Program", "2014");
14
15    // campos de instância
16    private final String title; // título de livro
17    private final String copyrightYear; // ano dos direitos autorais
18
19    // construtor enum
20    Book(String title, String copyrightYear)
21    {
22        this.title = title;
23        this.copyrightYear = copyrightYear;
24    }
}
```

8.9 Tipos enum

```
25      // acessor para título de campo
26      public String getTitle()
27      {
28          return title;
29      }
30
31
32      // acessor para o campo copyrightYear
33      public String getCopyrightYear()
34      {
35          return copyrightYear;
36      }
37 } // fim do enum Book
```

8.9 Tipos enum

```
1 // Figura 8.11: EnumTest.java
2 // Testando o tipo enum Book.
3 import java.util.EnumSet;
4
5 public class EnumTest
6 {
7     public static void main(String[] args)
8     {
9         System.out.println("All books:");
10
11         // imprime todos os livros em enum Book
12         for (Book book : Book.values())
13             System.out.printf("%-10s%-45s%s%n", book,
14                               book.getTitle(),book.getCopyrightYear());
15
16         System.out.printf("%nDisplay a range of enum constants:%n");
17
18         // imprime os primeiros quatro livros
19         for (Book book : EnumSet.range(Book.JHTP, Book.CPPHTP))
20             System.out.printf("%-10s%-45s%s%n", book,
21                               book.getTitle(),book.getCopyrightYear());
22     }
23 } // fim da classe EnumTest
```

8.9 Tipos enum

All books:

JHTP	Java How to Program	2015
CHTP	C How to Program	2013
IW3HTP	Internet & World Wide Web How to Program	2012
CPPHTP	C++ How to Program	2014
VBHTP	Visual Basic How to Program	2014
CSHARPHTP	Visual C# How to Program	2014

Display a range of enum constants:

JHTP	Java How to Program	2015
CHTP	C How to Program	2013
IW3HTP	Internet & World Wide Web How to Program	2012
CPPHTP	C++ How to Program	2014

8.10 Coleta de lixo

- A Java Virtual Machine (JVM) realiza a **coleta de lixo** automaticamente para reivindicar a memória ocupada pelos objetos que não estão mais em uso. Quando não há mais referências a um objeto, ele é marcado para coleta de lixo.
- A memória desse objeto pode ser reivindicada quando a JVM executa seu **coletor de lixo**.

8.11 Membros da classe static

- Uma **variável static** representa informações por toda a classe, que são **compartilhadas** entre os objetos da classe.
- Variáveis static têm **escopo de classe**. Os membros public static de uma classe podem ser acessados **por meio de uma referência a qualquer objeto da classe** ou qualificando o nome de membro com o nome de classe e um ponto (.).
- O código cliente só pode acessar os membros da classe static de uma classe private por meio dos métodos da classe.
- Membros da classe static existem assim que a classe é carregada na memória.

8.11 Membros da classe static

- Um método declarado `static` não pode acessar as variáveis de instância e os métodos de instância da classe, porque um método `static` **pode ser chamado mesmo quando nenhum objeto da classe foi instanciado.**
- A referência `this` não pode ser utilizada em um método `static`.

8.11 Membros da classe static

```
1 // Figura 8.12: Employee.java
2 // Variável static utilizada para manter uma contagem do número de
3 // objetos Employee na memória.
4
5 public class Employee
6 {
7     private static int count = 0; // número de Employees criados
8     private String firstName;
9     private String lastName;
10
11    // inicializa Employee, adiciona 1 a static count e
12    // gera a saída de String indicando que o construtor foi chamado
13    public Employee(String firstName, String lastName)
14    {
15        this.firstName = firstName;
16        this.lastName = lastName;
17
18        ++count; // incrementa contagem estática de empregados
19        System.out.printf("Employee constructor: %s %s; count = %d%n",
20                          firstName, lastName, count);
21    }
22
23    // obtém o primeiro nome
24    public String getFirstName()
25    {
26        return firstName;
27    }
28
29    // obtém o último nome
```

continua

8.11 Membros da classe static

```
30     public String getLastName()
31     {
32         return lastName;
33     }
34
35     // método estático para obter valor de contagem de estática
36     public static int getCount()
37     {
38         return count;
39     }
40 } // fim da classe Employee
```

8.11 Membros da classe static

```
1 // Figura 8.13: EmployeeTest.java
2 // Demonstração do membro static.
3
4 public class EmployeeTest
5 {
6     public static void main(String[] args)
7     {
8         // mostra que a contagem é 0 antes de criar Employees
9         System.out.printf("Employees before instantiation: %d%n",
10                         Employee.getCount());
11
12         // cria dois Employees; a contagem deve ser 2
13         Employee e1 = new Employee("Susan", "Baker");
14         Employee e2 = new Employee("Bob", "Blue");
15
16         // mostra que a contagem é 2 depois de criar dois Employees
17         System.out.printf("%nEmployees after instantiation:%n");
18         System.out.printf("via e1.getCount(): %d%n", e1.getCount());
19         System.out.printf("via e2.getCount(): %d%n", e2.getCount());
20         System.out.printf("via Employee.getCount(): %d%n",
21                         Employee.getCount());
22
23         // obtém nomes de Employees
24         System.out.printf("%nEmployee 1: %s %s%nEmployee 2: %s %s%n",
```

continua

8.11 Membros da classe static

```
25      e1.getFirstName(), e1.getLastName(),  
26          e2.getFirstName(), e2.getLastName());  
27      }  
28  } // fim da classe EmployeeTest
```

continuação

```
Employees before instantiation: 0  
Employee constructor: Susan Baker; count = 1  
Employee constructor: Bob Blue; count = 2  
Employees after instantiation:  
via e1.getCount(): 2  
via e2.getCount(): 2  
via Employee.getCount(): 2
```

```
Employee 1: Susan Baker  
Employee 2: Bob Blue
```

8.12 Importação static

- Uma declaração de **importação static** permite referenciar membros static importados sem o nome de classe e um ponto (.).
- Uma **única declaração de importação static** importa um membro static e uma **importação static por demanda** importa todos os membros static de uma classe.

8.12 Importação static

```
1 // Figura 8.14: StaticImportTest.java
2 // Importação static dos métodos da classe Math.
3 import static java.lang.Math.*;
4
5 public class StaticImportTest
6 {
7     public static void main(String[] args)
8     {
9         System.out.printf("sqrt(900.0) = %.1f%n", sqrt(900.0));
10    System.out.printf("ceil(-9.8) = %.1f%n", ceil(-9.8));
11    System.out.printf("E = %f%n", E);
12    System.out.printf("PI = %f%n", PI);
13 }
14 } // fim da classe StaticImportTest
```

```
sqrt(900.0) = 30.0
ceil(-9.8) = -9.0
E = 2.718282
PI = 3.141593
```

8.13 Variáveis de instância final

- No contexto de um aplicativo, o **princípio do menor privilégio** afirma que deve ser concedida ao código somente a quantidade de privilégio e acesso que ele precisa para realizar sua tarefa designada.
- A palavra-chave `final` especifica que uma variável não é modificável. Essas variáveis devem ser inicializadas quando são declaradas ou por cada um dos construtores de uma classe.

8.14 Acesso de pacote

- Se nenhum modificador de acesso for especificado para um método ou variável quando esse método ou variável é declarado em uma classe, o método ou variável é considerado como tendo **acesso de pacote**.

8.14 Acesso de pacote

```
1 // Figura 8.15: PackageDataTest.java
2 // Membros de acesso de pacote de uma classe permanecem acessíveis a outras classes
3 // no mesmo pacote.
4
5 public class PackageDataTest
6 {
7     public static void main(String[] args)
8     {
9         PackageData packageData = new PackageData();
10
11         // gera saída da representação String de packageData
12         System.out.printf("After instantiation:%n%s%n", packageData);
13
14         // muda os dados de acesso de pacote no objeto packageData
15         packageData.number = 77;
16         packageData.string = "Goodbye";
17
18         // gera saída da representação String de packageData
19         System.out.printf("%nAfter changing values:%n%s%n", packageData);
20     }
21 } // fim da classe PackageDataTest
```

8.14 Acesso de pacote

```
22 // classe com variáveis de instância de acesso de pacote
23 class PackageData
24 {
25     int number; // variável de instância de acesso de pacote
26     String string; // variável de instância de acesso de pacote
27
28     // construtor
29     public PackageData()
30     {
31         number = 0;
32         string = "Hello";
33     }
34
35
36     // retorna a representação String do objeto PackageData
37     public String toString()
38     {
39         return String.format("number: %d; string: %s", number, string);
40     }
41 } // fim da classe PackageData
```

continuação

After instantiation:
number: 0; string: Hello

After changing values:
number: 77; string: Goodbye

Referências Bibliográficas

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