

# Java: como programar

## Programação Orientada à Objetos

- Douglas Baptista de Godoy

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

 [github.com/douglasbgodoy](https://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.

## Estudo de caso da classe `Time`

- Os métodos `public` de uma classe também são conhecidos como os **serviços `public`** ou **interface `public`** da classe.
- Eles apresentam aos clientes da classe uma visualização dos **serviços fornecidos**.

## Estudo de caso da classe `Time`

- Membros `private` de uma classe não são acessíveis aos clientes.
- O método `static format` da classe `String` é semelhante ao método `System.out.printf`, exceto que `format` retorna uma `String` formatada em vez de exibi-la em uma janela de comando.

## Operador condicional (?:)

O Java fornece o **operador condicional (?:)**, que pode ser utilizado no lugar de uma instrução `if...else`. Isso pode tornar o código mais curto e mais claro. O operador condicional é o único **operador ternário** do Java (isto é, um operador que recebe *três* operandos). Juntos, os operandos e o símbolo `?:` formam uma **expressão condicional**. O primeiro operando (à esquerda do `?`) é uma **expressão boolean** (isto é, uma *condição* que é avaliada como um valor boolean — **true** ou **false**), o segundo operando (entre o `?` e `:`) é o valor da expressão condicional se a expressão boolean for **true** e o terceiro operando (à direita do `:`) é o valor da expressão condicional se a expressão boolean for avaliada como **false**. Por exemplo, a instrução

```
System.out.println(studentGrade >= 60 ? "Passed" : "Failed");
```

imprime o valor do argumento da expressão condicional de `println`. A expressão condicional nessa instrução é avaliada para a string `"Passed"` se a expressão boolean `studentGrade >= 60` for verdadeira e para a string `"Failed"` se a expressão boolean for falsa. Portanto, essa instrução com o operador condicional realiza essencialmente a mesma função da instrução `if...else` mostrada anteriormente nesta seção. A precedência do operador condicional é baixa, então a expressão condicional inteira normalmente é colocada entre parênteses. Veremos que as expressões condicionais podem ser utilizadas em algumas situações nas quais as instruções `if...else` não podem.

## 8.2 Estudo de caso da classe Time

```
1 // Figura 8.1: Time1.java
2 // Declaração de classe Time1 mantém a hora no formato de 24 horas.
3
4 public class Time1
5 {
6     private int hour; // 0 - 23
7     private int minute; // 0 - 59
8     private int second; // 0 - 59
9
10    // configura um novo valor de hora usando formato universal;
11    // assegura que os dados permaneçam consistentes configurando valores inválidos
12    // como zero
13    public void setTime( int h, int m, int s )
14    {
15        hour = ( ( h >= 0 && h < 24 ) ? h : 0 ); // valida horas
16        minute = ( ( m >= 0 && m < 60 ) ? m : 0 ); // valida minutos
17        second = ( ( s >= 0 && s < 60 ) ? s : 0 ); // valida segundos
18    } // fim do método setTime
19
20    // converte em String no formato de hora universal (HH:MM:SS)
21    public String toUniversalString()
22    {
23        return String.format( "%02d:%02d:%02d", hour, minute, second );
24    } // fim do método do toUniversalString
```

Variáveis de instância representam a hora no formato de 24 horas

Valida os valores iniciais de hora, minuto e segundo

Formato de hora de 24 horas

**Figura 8.1** | Declaração da classe Time1 mantém a hora no formato de 24 horas. (Parte 1 de 2.)

## 8.2 Estudo de caso da classe Time

```
25 // converte em String no formato padrão hora (H:MM:SS AM ou PM)
26 public String toString()
27 {
28     return String.format( "%d:%02d:%02d %s",
29         ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 ),
30         minute, second, ( hour < 12 ? "AM" : "PM" ) );
31 } // fim do método toString
32 } // fim da classe Time1
```

Formata a hora no formato de hora de 12 horas; esse também é o formato String padrão para Time1

**Figura 8.1** | Declaração da classe Time1 mantém a hora no formato de 24 horas. (Parte 2 de 2.)

## 8.2 Estudo de caso da classe Time

```
1 // Figura 8.2: Time1Test.java
2 // objeto Time1 utilizado em um aplicativo.
3
4 public class Time1Test
5 {
6     public static void main( String[] args )
7     {
8         // cria e inicializa um objeto Time1
9         Time1 time = new Time1(); // invoca o construtor Time1
10
11         // gera saída de representações de string da hora
12         System.out.print( "The initial universal time is: " );
13         System.out.println( time.toUniversalString() );
14         System.out.print( "The initial standard time is: " );
15         System.out.println( time.toString() );
16         System.out.println(); // gera saída de uma linha em branco
17
18         // altera a hora e gera saída da hora atualizada
19         time.setTime( 13, 27, 6 );
20         System.out.print( "Universal time after setTime is: " );
21         System.out.println( time.toUniversalString() );
22         System.out.print( "Standard time after setTime is: " );
23         System.out.println( time.toString() );
24         System.out.println(); // gera saída de uma linha em branco
```

← Cria o objeto Time1 padrão

← Obtém a representação String da hora no formato de 24 horas

← Obtém a representação String da hora no formato de 12 horas

← Configura a hora usando valores válidos para hora, minuto e segundo

**Figura 8.2** | Objeto Time1 utilizado em um aplicativo. (Parte I de 2.)



## 8.2 Estudo de caso da classe Time

```
25
26 // configura a hora com valores inválidos; gera saída da hora atualizada
27 time.setTime( 99, 99, 99 );
28 System.out.println( "After attempting invalid settings:" );
29 System.out.print( "Universal time: " );
30 System.out.println( time.toUniversalString() );
31 System.out.print( "Standard time: " );
32 System.out.println( time.toString() );
33 } // fim de main
34 } // fim da classe Time1Test
```

Configura a hora usando valores inválidos para hora, minuto e segundo

```
The initial universal time is: 00:00:00
The initial standard time is: 12:00:00 AM

Universal time after setTime is: 13:27:06
Standard time after setTime is: 1:27:06 PM

After attempting invalid settings:
Universal time: 00:00:00
Standard time: 12:00:00 AM
```

**Figura 8.2** | Objeto Time1 utilizado em um aplicativo. (Parte 2 de 2.)

## 8.2 Estudo de caso da classe Time

Visão sob outra perspectiva.

## 8.2 Estudo de caso da classe Time

---

```
1 // Figura 8.1: Time1.java
2 // Declaração de classe Time1 mantém a data/hora no formato de 24 horas.
3
4 public class Time1
5 {
6     private int hour; // 0 - 23
7     private int minute; // 0 - 59
8     private int second; // 0 - 59
9
10    // configura um novo valor de tempo usando hora universal; lança uma
11    // exceção se a hora, minuto ou segundo for inválido
12    public void setTime(int hour, int minute, int second)
13    {
14        // valida hora, minuto e segundo
15        if (hour < 0 || hour >= 24 || minute < 0 || minute >= 60 ||
16            second < 0 || second >= 60)
```

*continua*

## 8.2 Estudo de caso da classe Time

continuação

```
17     {
18         throw new IllegalArgumentException(
19             "hour, minute and/or second was out of range");
20     }
21
22     this.hour = hour;
23     this.minute = minute;
24     this.second = second;
25 }
26
27 // converte em String no formato de data/hora universal (HH:MM:SS)
28 public String toUniversalString()
29 {
30     return String.format("%02d:%02d:%02d", hour, minute, second);
31 }
32
33 // converte em String no formato padrão de data/hora (H:MM:SS AM ou PM)
34 public String toString()
35 {
36     return String.format("%d:%02d:%02d %s",
37         ((hour == 0 || hour == 12) ? 12 : hour % 12),
38         minute, second, (hour < 12 ? "AM" : "PM"));
39 }
40 } // fim da classe Time1
```

**Figura 8.1** | Declaração da classe Time1 mantém a data/hora no formato de 24 horas.

## 8.2 Estudo de caso da classe Time

```
1 // Figura 8.2: Time1Test.java
2 // objeto Time1 utilizado em um aplicativo.
3
4 public class Time1Test
5 {
6     public static void main(String[] args)
7     {
8         // cria e inicializa um objeto Time1
9         Time1 time = new Time1(); // invoca o construtor Time1
10
11         // gera saída de representações de string da data/hora
12         displayTime("After time object is created", time);
13         System.out.println();
14
15         // altera a data/hora e gera saída da data/hora atualizada
16         time.setTime(13, 27, 6);
17         displayTime("After calling setTime", time);
18         System.out.println();
19
20         // tenta definir data/hora com valores inválidos
21         try
22         {
23             time.setTime(99, 99, 99); // todos os valores fora do intervalo
24         }
25         catch (IllegalArgumentException e)
26         {
27             System.out.printf("Exception: %s\n\n", e.getMessage());
28         }
29     }
30 }
```

*continua*

## 8.2 Estudo de caso da classe Time

continuação

```
29
30     // exibe a data/hora após uma tentativa de definir valores inválidos
31     displayTime("After calling setTime with invalid values", time);
32 }
33
34 // exibe um objeto Time1 nos formatos de 24 horas e 12 horas
35 private static void displayTime(String header, Time1 t)
36 {
37     System.out.printf("%s\nUniversal time: %s\nStandard time: %s\n",
38         header, t.toUniversalString(), t.toString());
39 }
40 } // fim da classe Time1Test
```

After time object is created  
Universal time: 00:00:00  
Standard time: 12:00:00 AM

After calling setTime  
Universal time: 13:27:06  
Standard time: 1:27:06 PM

Exception: hour, minute and/or second was out of range

After calling setTime with invalid values  
Universal time: 13:27:06  
Standard time: 1:27:06 PM

**Figura 8.2** | Objeto Time1 utilizado em um aplicativo.

## 8.4 Referenciando membros do objeto atual com a referência this

## 8.4 Referenciando membros do objeto atual com a referência this

---

```
1  // Figura 8.4: ThisTest.java
2  // this usado implícita e explicitamente para referenciar membros de um objeto.
3
4  public class ThisTest
5  {
6      public static void main( String[] args )
7      {
8          SimpleTime time = new SimpleTime( 15, 30, 19 );
9          System.out.println( time.buildString() );
10     } // fim de main
11 } // fim da classe ThisTest
12
13 // classe SimpleTime demonstra a referência "this"
14 class SimpleTime
15 {
16     private int hour; // 0-23
17     private int minute; // 0-59
18     private int second; // 0-59
19 }
```

---

**Figura 8.4** | this usado implícita e explicitamente como uma referência a membros de um objeto. (Parte I de 3)



## 8.4 Referenciando membros do objeto atual com a referência `this`

```
20 // se o construtor utilizar nomes de parâmetro idênticos a
21 // nomes de variáveis de instância a referência "this" será
22 // exigida para distinguir entre nomes
23 public SimpleTime( int hour, int minute, int second )
24 {
25     this.hour = hour; // configura a hora do objeto "this"
26     this.minute = minute; // configura os minutos do objeto "this"
27     this.second = second; // configura os segundos do objeto "this"
28 } // fim do construtor SimpleTime
29
30 // usa "this" explícito e implícito para chamar toUniversalString
31 public String buildString()
32 {
33     return String.format( "%24s: %s\n%24s: %s",
34         "this.toUniversalString()", this.toUniversalString(),
35         "toUniversalString()", toUniversalString() );
36 } // fim do método buildString
37
```

A referência `this` permite acessar explicitamente variáveis de instância quando elas estão sombreadas pelas variáveis locais de mesmo nome

A referência `this` não é exigida para chamar outros métodos da mesma classe

**Figura 8.4** | `this` usado implícita e explicitamente como uma referência a membros de um objeto. (Parte I de 3)

## 8.4 Referenciando membros do objeto atual com a referência this

```
38 // converte em String no formato de hora universal (HH:MM:SS)
39 public String toUniversalString()
40 {
41     // "this" não é requerido aqui para acessar variáveis de instância,
42     // porque o método não tem variáveis locais com os mesmos
43     // nomes das variáveis de instância
44     return String.format( "%02d:%02d:%02d",
45         this.hour, this.minute, this.second );
46 } // fim do método do toUniversalString
47 } // fim da classe SimpleTime
```

"this" não é requerido aqui uma vez que as variáveis de instância não estão sombreadas

```
this.toUniversalString(): 15:30:19
toUniversalString(): 15:30:19
```

**Figura 8.4** | this usado implícita e explicitamente como uma referência a membros de um objeto. (Parte 3 de 3)

## 8.5 Estudo de caso da classe Time: construtores sobrecarregados

## 8.5 Estudo de caso da classe Time: construtores sobrecarregados

```
1 // Figura 8.5: Time2.java
2 // declaração da classe Time2 com construtores sobrecarregados.
3
4 public class Time2
5 {
6     private int hour; // 0 - 23
7     private int minute; // 0 - 59
8     private int second; // 0 - 59
9
10    // construtor sem argumento Time2 : inicializa cada variável de instância
11    // com zero; assegura que objetos Time2 iniciam em um estado consistente
12    public Time2()
13    {
14        this( 0, 0, 0 ); // invoca o construtor Time2 com três argumentos
15    } // fim do construtor sem argumento Time2
16
17    // Construtor Time2: hour fornecido, min e sec padronizados como 0
18    public Time2( int h )
19    {
20        this( h, 0, 0 ); // invoca o construtor Time2 com três argumentos
21    } // fim do construtor de um argumento Time2
22
```

Invoca o construtor de três argumentos

Invoca o construtor de três argumentos

**Figura 8.5** | Classe Time2 com construtores sobrecarregados. (Parte I de 5.)

## 8.5 Estudo de caso da classe Time: construtores sobrecarregados

```
23 // Construtor Time2: hour e min fornecidos, sec padronizado como 0
24 public Time2( int h, int m )
25 {
26     this( h, m, 0 ); // invoca o construtor Time2 com três argumentos
27 } // fim do construtor de dois argumentos Time2
28
29 // Construtor Time2: hour, min e sec fornecidos
30 public Time2( int h, int m, int s )
31 {
32     setTime( h, m, s ); // invoca setTime para validar a hora
33 } // fim do construtor de três argumentos Time2
34
35 // Construtor Time2: outro objeto Time2 fornecido
36 public Time2( Time2 time )
37 {
38     // invoca o construtor de três argumentos Time2
39     this( time.getHour(), time.getMinute(), time.getSecond() );
40 } // fim do construtor Time2 com um argumento de objeto Time2
41
```

Invoca o construtor de três argumentos

Invoca setTime para validar os dados

Invoca o construtor de três argumentos

**Figura 8.5** | Classe Time2 com construtores sobrecarregados. (Parte 2 de 5.)

## 8.5 Estudo de caso da classe Time: construtores sobrecarregados

```
42 // Métodos set
43 // configura um novo valor de hora usando o formato universal;
44 // assegura que os dados permaneçam consistentes configurando
44 // valores inválidos como zero
45 public void setTime( int h, int m, int s )
46 {
47     setHour( h ); // configura hour
48     setMinute( m ); // configura minute
49     setSecond( s ); // configura second
50 } // fim do método setTime
51
52 // valida e configura a hora
53 public void setHour( int h )
54 {
55     hour = ( ( h >= 0 && h < 24 ) ? h : 0 );
56 } // fim do método setHour
57
58 // valida e configura os minutos
59 public void setMinute( int m )
60 {
61     minute = ( ( m >= 0 && m < 60 ) ? m : 0 );
62 } // fim do método setMinute
63
```

**Figura 8.5** | Classe Time2 com construtores sobrecarregados. (Parte 3 de 5.)

## 8.5 Estudo de caso da classe Time: construtores sobrecarregados

```
64 // valida e configura os segundos
65 public void setSecond( int s )
66 {
67     second = ( ( s >= 0 && s < 60 ) ? s : 0 );
68 } // fim do método setSecond
69
70 // Métodos get
71 // obtém valor da hora
72 public int getHour()
73 {
74     return hour;
75 } // fim do método getHour
76
77 // obtém valor dos minutos
78 public int getMinute()
79 {
80     return minute;
81 } // fim do método getMinute
82
83 // obtém valor dos segundos
84 public int getSecond()
85 {
86     return second;
87 } // fim do método getSecond
```

**Figura 8.5** | Classe Time2 com construtores sobrecarregados. (Parte 4 de 5.)

## 8.5 Estudo de caso da classe Time: construtores sobrecarregados

---

```
88
89 // converte em String no formato de hora universal (HH:MM:SS)
90 public String toUniversalString()
91 {
92     return String.format(
93         "%02d:%02d:%02d", getHour(), getMinute(), getSecond() );
94 } // fim do método do toUniversalString
95
96 // converte em String no formato padrão de data (H:MM:SS AM ou PM)
97 public String toString()
98 {
99     return String.format( "%d:%02d:%02d %s",
100         ( (getHour() == 0 || getHour() == 12) ? 12 : getHour() % 12 ),
101         getMinute(), getSecond(), ( getHour() < 12 ? "AM" : "PM" ) );
102 } // fim do método toString
103 } // fim da classe Time2
```

---

**Figura 8.5** | Classe Time2 com construtores sobrecarregados. (Parte 5 de 5.)



## 8.5 Estudo de caso da classe Time: construtores sobrecarregados

```
1 // Figura 8.6: Time2Test.java
2 // Construtores sobrecarregados utilizados para inicializar objetos Time2.
3
4 public class Time2Test
5 {
6     public static void main( String[] args )
7     {
8         Time2 t1 = new Time2(); // 00:00:00
9         Time2 t2 = new Time2( 2 ); // 02:00:00
10        Time2 t3 = new Time2( 21, 34 ); // 21:34:00
11        Time2 t4 = new Time2( 12, 25, 42 ); // 12:25:42
12        Time2 t5 = new Time2( 27, 74, 99 ); // 00:00:00
13        Time2 t6 = new Time2( t4 ); // 12:25:42
14
15        System.out.println( "Constructed with:" );
16        System.out.println( "t1: all arguments defaulted" );
17        System.out.printf( "    %s\n", t1.toUniversalString() );
18        System.out.printf( "    %s\n", t1.toString() );
19
20        System.out.println(
21            "t2: hour specified; minute and second defaulted" );
22        System.out.printf( "    %s\n", t2.toUniversalString() );
23        System.out.printf( "    %s\n", t2.toString() );
24    }
}
```

O compilador determina qual construtor chamar com base no número e nos tipos dos argumentos

**Figura 8.6** | Construtores sobrecarregados utilizados para inicializar objetos Time2. (Parte I de 3.)

## 8.5 Estudo de caso da classe Time: construtores sobrecarregados

---

```
25     System.out.println(  
26         "t3: hour and minute specified; second defaulted" );  
27     System.out.printf( "    %s\n", t3.toUniversalString() );  
28     System.out.printf( "    %s\n", t3.toString() );  
29  
30     System.out.println( "t4: hour, minute and second specified" );  
31     System.out.printf( "    %s\n", t4.toUniversalString() );  
32     System.out.printf( "    %s\n", t4.toString() );  
33  
34     System.out.println( "t5: all invalid values specified" );  
35     System.out.printf( "    %s\n", t5.toUniversalString() );  
36     System.out.printf( "    %s\n", t5.toString() );  
37  
38     System.out.println( "t6: Time2 object t4 specified" );  
39     System.out.printf( "    %s\n", t6.toUniversalString() );  
40     System.out.printf( "    %s\n", t6.toString() );  
41 } // fim de main  
42 } // fim da classe Time2Test
```

---

**Figura 8.6** | Construtores sobrecarregados utilizados para inicializar objetos Time2. (Parte 2 de 3.)

## 8.5 Estudo de caso da classe Time: construtores sobrecarregados

```
t1: all arguments defaulted
    00:00:00
    12:00:00 AM
t2: hour specified; minute and second defaulted
    02:00:00
    2:00:00 AM
t3: hour and minute specified; second defaulted
    21:34:00
    9:34:00 PM
t4: hour, minute and second specified
    12:25:42
    12:25:42 PM
t5: all invalid values specified
    00:00:00
    12:00:00 AM
t6: Time2 object t4 specified
    12:25:42
    12:25:42 PM
```

**Figura 8.6** | Construtores sobrecarregados utilizados para inicializar objetos Time2. (Parte 3 de 3.)

## Referências Bibliográficas

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