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)
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
```

```
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          // 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
```

8.3 Controlando o acesso a membros

```
1  // Figura 8.3: MemberAccessTest.java
2  // Membros privados da classe Time1 não são acessíveis.
3  public class MemberAccessTest
4  {
5     public static void main(String[] args)
6     {
7        Time1 time = new Time1(); // cria e inicializa o objeto Time1
8
9        time.hour = 7; // erro: hour tem acesso privado em Time1
10       time.minute = 15; // erro: minute tem acesso privado em Time1
11       time.second = 30; // erro: second tem acesso privado em Time1
12    }
13 } // fim da classe MemberAccessTest
```

```
MemberAccessTest.java:9: hour tem acesso privado em Time1
     time.hour = 7; // erro: hour tem acesso privado em Time1
        ^
MemberAccessTest.java:10: minute tem acesso privado em Time1
     time.minute = 15; // erro: minute tem acesso privado em Time1
        ^
MemberAccessTest.java:11: second tem acesso privado em Time1
     time.second = 30; // erro: second tem acesso privado em Time1
        ^
3 errors
```

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

```
1  // Figura 8.4: ThisTest.java
2  // this utilizado implícita e explicitamente para referência a 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    }
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
```

```
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 os 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 o minuto do objeto "this"
27         this.second = second; // configura o segundo do objeto "this"
28      }
29
30      // utilizam "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      }
37
38      // converte em String no formato de data/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      }
47   } // fim da classe SimpleTime
```

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

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:
11     // inicializa cada variável de instância para zero
12     public Time2()
13     {
14        this(0, 0, 0); // invoca o construtor com três argumentos
15     }
16
17     // Construtor Time2: hora fornecida, minuto e segundo padronizados para 0
18     public Time2(int hour)
19     {
20        this(hour, 0, 0); // invoca o construtor com três argumentos
21     }
22
23     // Construtor Time2: hora e minuto fornecidos, segundo padronizado para 0
24     public Time2(int hour, int minute)
25     {
26        this(hour, minute, 0); // invoca o construtor com três argumentos
27     }
28
29     // Construtor Time2: hour, minute e second fornecidos
30     public Time2(int hour, int minute, int second)
31     {
32        if (hour < 0 || hour >= 24)
33           throw new IllegalArgumentException("hour must be 0-23");
34
35        if (minute < 0 || minute >= 60)
36           throw new IllegalArgumentException("minute must be 0-59");
37
38        if (second < 0 || second >= 60)
39           throw new IllegalArgumentException("second must be 0-59");
40
```

```
41          this.hour = hour;
42          this.minute = minute;
43          this.second = second;
44       }
45
46       // Construtor Time2: outro objeto Time2 fornecido
47       public Time2(Time2 time)
48       {
49          // invoca o construtor com três argumentos
50          this(time.getHour(), time.getMinute(), time.getSecond());
51       }
52
53       // Métodos set
54       // Configura um novo valor de tempo usando hora universal;
55       // valida os dados
56       public void setTime(int hour, int minute, int second)
57       {
58          if (hour < 0 || hour >= 24)
59             throw new IllegalArgumentException("hour must be 0-23");
60
61          if (minute < 0 || minute >= 60)
62             throw new IllegalArgumentException("minute must be 0-59");
63
64          if (second < 0 || second >= 60)
65             throw new IllegalArgumentException("second must be 0-59");
66
67          this.hour = hour;
68          this.minute = minute;
69          this.second = second;
70       }
71
72       // valida e configura a hora
73       public void setHour(int hour)
74       {
75          if (hour < 0 || hour >= 24)
76             throw new IllegalArgumentException("hour must be 0-23");
77
78          this.hour = hour;
79       }
80
81       // valida e configura os minutos
82       public void setMinute(int minute)
83       {
84          if (minute < 0 || minute >= 60)
85             throw new IllegalArgumentException("minute must be 0-59");
86
87          this.minute = minute;
88       }
89
90       // valida e configura os segundos
91       public void setSecond(int second)
92       {
93          if (second < 0 || second >= 60)
94             throw new IllegalArgumentException("second must be 0-59");
95
96          this.second = second;
97       }
98
99       // Métodos get
100      // obtém valor da hora
101      public int getHour()
102      {
103         return hour;
104      }
105
106      // obtém valor dos minutos
107      public int getMinute()
108      {
109         return minute;
110      }
111
112      // obtém valor dos segundos
113      public int getSecond()
114      {
115         return second;
116      }
117
```

```
118      // converte em String no formato de data/hora universal (HH:MM:SS)
119      public String toUniversalString()
120      {
121         return String.format(
122            "%02d:%02d:%02d", getHour(), getMinute(), getSecond());
123      }
124
125      // converte em String no formato padrão de data/hora (H:MM:SS AM ou PM)
126      public String toString()
127      {
128         return String.format("%d:%02d:%02d %s",
129            ((getHour() == 0 || getHour() == 12) ? 12 : getHour() % 12),
130            getMinute(), getSecond(), (getHour() < 12 ? "AM" : "PM"));
131      }
132   } // fim da classe Time2
```

```
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(t4); // 12:25:42
13
14         System.out.println("Constructed with:");
15         displayTime("t1: all default arguments", t1);
16         displayTime("t2: hour specified; default minute and second", t2);
17         displayTime("t3: hour and minute specified; default second", t3);
18         displayTime("t4: hour, minute and second specified", t4);
19         displayTime("t5: Time2 object t4 specified", t5);
20
21         // tenta inicializar t6 com valores inválidos
22         try
23         {
24            Time2 t6 = new Time2(27, 74, 99); // valores inválidos
25         }
26         catch (IllegalArgumentException e)
27         {
28            System.out.printf("%nException while initializing t6: %s%n",
29               e.getMessage());
30         }
31      }
32
33      // exibe um objeto Time2 nos formatos de 24 horas e 12 horas
34      private static void displayTime(String header, Time2 t)
35      {
36         System.out.printf("%s%n   %s%n   %s%n",
37            header, t.toUniversalString(), t.toString());
38      }
39   } // fim da classe Time2Test
```

```
Constructed with:
t1: all default arguments
   00:00:00
   12:00:00 AM
t2: hour specified; default minute and second
   02:00:00
   2:00:00 AM
t3: hour and minute specified; default second
   21:34:00
   9:34:00 PM
t4: hour, minute and second specified
   12:25:42
   12:25:42 PM
t5: Time2 object t4 specified
   12:25:42
   12:25:42 PM
Exception while initializing t6: hour must be 0-23
```

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    {
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
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
```

```
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;
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
```

```
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

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

```java
// Figura 8.11: EnumTest.java
// Testando o tipo enum Book.
import java.util.EnumSet;

public class EnumTest
{
   public static void main(String[] args)
   {
      System.out.println("All books:");

      // imprime todos os livros em enum Book
      for (Book book : Book.values())
         System.out.printf("%-10s%-45s%s%n", book,
             book.getTitle(),book.getCopyrightYear());

      System.out.printf("%nDisplay a range of enum constants:%n");

      // imprime os primeiros quatro livros
      for (Book book : EnumSet.range(Book.JHTP, Book.CPPHTP))
         System.out.printf("%-10s%-45s%s%n", book,
             book.getTitle(),book.getCopyrightYear());
   }
} // fim da classe EnumTest
```

```
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
```

8.11 Membros da classe `static`

```java
// Figura 8.12: Employee.java
// Variável static utilizada para manter uma contagem do número de
// objetos Employee na memória.

public class Employee
{
   private static int count = 0; // número de Employees criados
   private String firstName;
   private String lastName;

   // inicializa Employee, adiciona 1 a static count e
   // gera a saída de String indicando que o construtor foi chamado
   public Employee(String firstName, String lastName)
   {
      this.firstName = firstName;
      this.lastName = lastName;

      ++count;  // incrementa contagem estática de empregados
      System.out.printf("Employee constructor: %s %s; count = %d%n",
         firstName, lastName, count);
   }

   // obtém o primeiro nome
   public String getFirstName()
   {
      return firstName;
   }

   // obtém o último nome
   public String getLastName()
   {
      return lastName;
   }

```

```
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
```

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

```
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`

Importando formulários `static`

Uma declaração de importação `static` tem duas formas — uma que importa um membro `static` particular (conhecido como **importação `static` simples**) e outra que importa *todos* os membros `static` de uma classe (conhecido como **importação `static` por demanda**). A sintaxe a seguir importa um membro `static` particular:

```
import static nomeDoPacote.NomeDaClasse.nomeDoMembroStatic;
```

onde *nomeDoPacote* é o pacote da classe (por exemplo, `java.lang`), *NomeDaClasse* é o nome da classe (por exemplo, `Math`) e *nomeDoMembroStatic* é o nome do campo ou método `static` (por exemplo, `PI` ou `abs`). A sintaxe a seguir importa *todos* os membros `static` de uma classe:

```
import static nomeDoPacote.NomeDaClasse.*;
```

```java
// Figura 8.14: StaticImportTest.java
// Importação static dos métodos da classe Math.
import static java.lang.Math.*;

public class StaticImportTest
{
   public static void main(String[] args)
   {
      System.out.printf("sqrt(900.0) = %.1f%n", sqrt(900.0));
      System.out.printf("ceil(-9.8) = %.1f%n", ceil(-9.8));
      System.out.printf("E = %f%n", E);
      System.out.printf("PI = %f%n", PI);
   }
} // fim da classe StaticImportTest
```

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

8.14 Acesso de pacote

```java
// Figura 8.15: PackageDataTest.java
// Membros de acesso de pacote de uma classe permanecem acessíveis a outras classes
// no mesmo pacote.

public class PackageDataTest
{
   public static void main(String[] args)
   {
      PackageData packageData = new PackageData();

      // gera saída da representação String de packageData
      System.out.printf("After instantiation:%n%s%n", packageData);

      // muda os dados de acesso de pacote no objeto packageData
      packageData.number = 77;
      packageData.string = "Goodbye";

      // gera saída da representação String de packageData
      System.out.printf("%nAfter changing values:%n%s%n", packageData);
   }
} // fim da classe PackageDataTest

// classe com variáveis de instância de acesso de pacote
class PackageData
{
   int number; // variável de instância de acesso de pacote
   String string; // variável de instância de acesso de pacote

   // construtor
   public PackageData()
   {
      number = 0;
      string = "Hello";
   }

   // retorna a representação String do objeto PackageData
   public String toString()
   {
      return String.format("number: %d; string: %s", number, string);
   }
} // fim da classe PackageData
```

```
After instantiation:
number: 0; string: Hello

After changing values:
number: 77; string: Goodbye
```

8.15 Usando `BigDecimal` para cálculos monetários precisos

```
1  // Interest.java
2  // Cálculos de juros compostos com BigDecimal.
3  import java.math.BigDecimal;
4  import java.text.NumberFormat;
5
6  public class Interest
7  {
8     public static void main(String args[])
9     {
10       // quantidade principal inicial antes dos juros
11       BigDecimal principal = BigDecimal.valueOf(1000.0);
12       BigDecimal rate = BigDecimal.valueOf(0.05); // taxa de juros
13
14       // exibe cabeçalhos
15       System.out.printf("%s%20s%n", "Year", "Amount on deposit");
16
17       // calcula quantidade de depósito para cada um dos dez anos
18       for (int year = 1; year <= 10; year++)
19       {
20          // calcula nova quantidade durante ano especificado
21          BigDecimal amount =
22             principal.multiply(rate.add(BigDecimal.ONE).pow(year));
23
24          // exibe o ano e a quantidade
25          System.out.printf("%4d%20s%n", year,
26             NumberFormat.getCurrencyInstance().format(amount));
27       }
28    }
29 } // fim da classe Interest
```

```
Year   Amount on deposit
   1         $1,050.00
   2         $1,102.50
   3         $1,157.62
   4         $1,215.51
   5         $1,276.28
   6         $1,340.10
   7         $1,407.10
   8         $1,477.46
   9         $1,551.33
  10         $1,628.89
```