9

Programação orientada a objetos: Herança

OBJETIVOS

- Neste capítulo, você aprenderá:
- Como a herança promove a capacidade de reutilização de software.
- As noções de superclasses e subclasses.
- Como utilizar a palavra-chave `extends` para criar uma classe que herda atributos e comportamentos de outra classe.
- Como utilizar o modificador de acesso `protected` para fornecer acesso de métodos de subclasse a membros de superclasse.
- Como acessar membros de superclasse com `super`.
- Como os construtores são utilizados em hierarquias de herança.
- Os métodos da classe `Object`, a superclasse direta ou indireta de todas as classes em Java.

9.1 Introdução

- **Herança:**
  - **Capacidade de reutilização de software.**
  - **Cria uma nova classe a partir de uma classe existente:**
    - **absorvendo os dados e comportamentos da classe existente; e**
    - **aprimorando-a com novas capacidades.**
  - **A subclasse estende a superclasse.**
    - **Subclasse:**
      - **Grupo mais especializado de objetos.**
      - **Comportamentos herdados da superclasse:**
        - **Podem se personalizar.**
      - **Comportamentos adicionais.**

9.1 Introdução (*Continuação*)

- **Hierarquia de classes.**
  - **Superclasse direta:**
    - **Herdada explicitamente (um nível acima na hierarquia).**
  - **Superclasse indireta:**
    - **Herdada de dois ou mais níveis acima na hierarquia.**
  - **Herança única:**
    - **Herda de uma superclasse.**
  - **Herança múltipla:**
    - **Herda de múltiplas superclasses.**
      - **O Java não suporta herança múltipla.**

9.2 Superclasses e subclasses

- **Superclasses e subclasses.**
  - **Freqüentemente, um objeto de uma classe também ‘é um’ objeto de uma outra classe.**
    - **Exemplo: Em geometria, um retângulo *é um* quadrilátero.**
      - **A classe `Retângulo` herda da classe `Quadrilátero`.**
      - **`Quadrilátero` : superclasse.**
      - **`Retângulo` : subclasse.**
  - **A superclasse em geral representa um conjunto maior de objetos do que as subclasses.**
    - **Exemplo:**
      - **Superclasse: `Veículo`**
        - **Carros, caminhões, barcos, bicicletas...**

Superclasse	Subclasses
Aluno	AlunoDeGraduação, AlunoDePósGraduação
Forma	Círculo, Triângulo, Retângulo
Financiamento	FinanciamentoDeCarro, FinanciamentoDeReformaDaCasa, FinanciamentoDeCasa
Empregado	CorpoDocente, Funcionários
ContaBancária	ContaCorrente, ContaDePoupança

Figura 9.1 | Exemplos de herança.

9.2 Superclasses e subclasses (*Continuação*)

- **Hierarquia de herança:**
  - **Relacionamentos de herança: estrutura de hierarquia do tipo árvore.**
  - **Cada classe torna-se:**
    - **Superclasse:**
      - **que fornece membros a outras classes.**
    - **OU**
    - **Subclasse:**
      - **que herda membros de outras classes.**

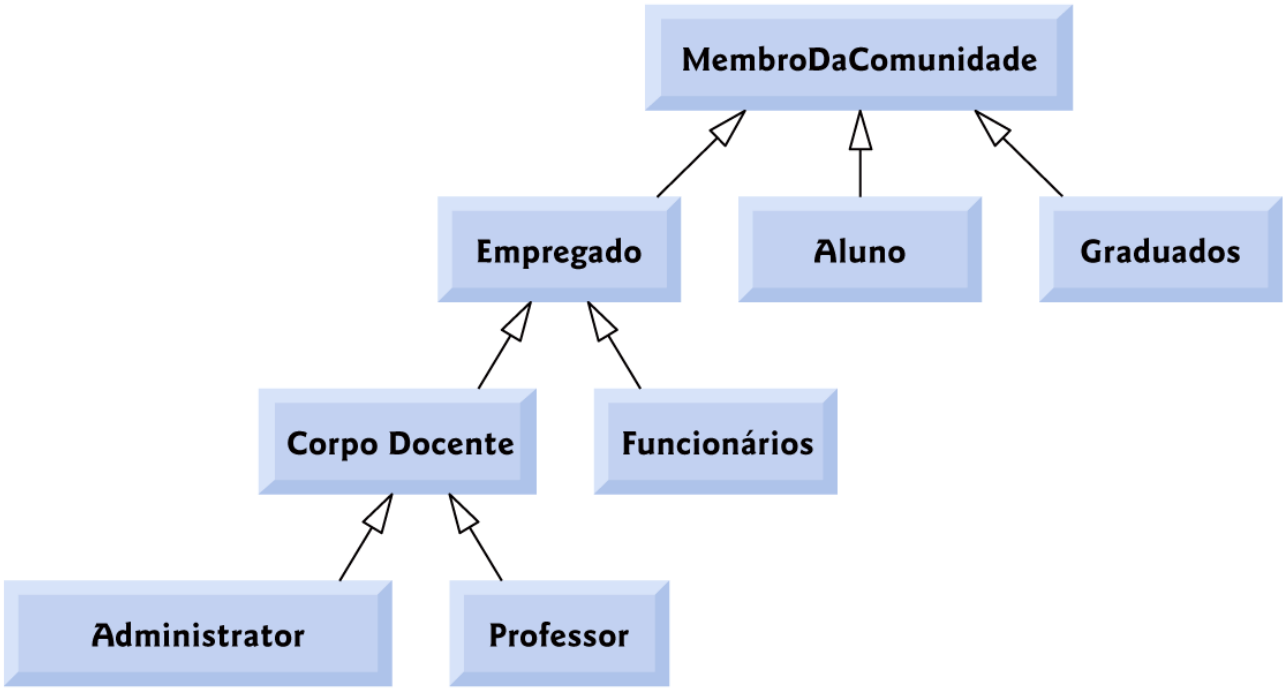

Figura 9.2 | Hierarquia de herança `MembrosDaComunidade` da universidade

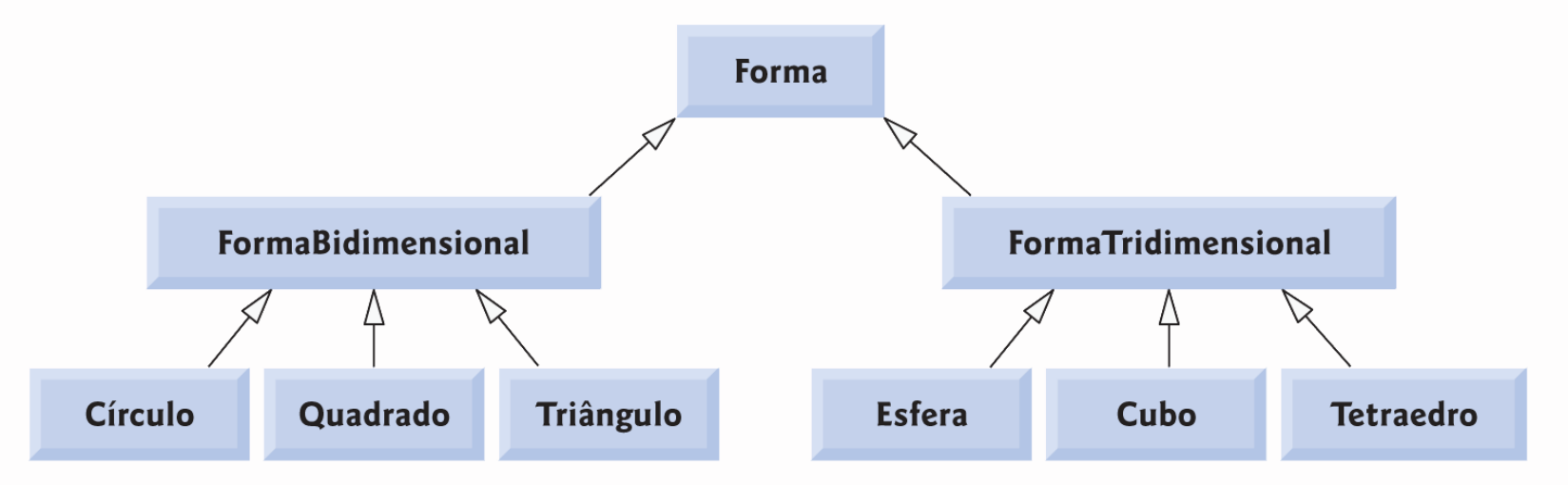

Figura 9.3 | Hierarquia de herança para Forma.

9.3 Membros `protected`

- Acesso `protected`:
  - Nível intermediário de proteção entre `public` e `private`.
  - Membros `protected` acessíveis pelos:
    - membros da superclasse;
    - membros da subclasse; e
    - membros da classe no mesmo pacote.
  - Acesso da subclasse ao membro da superclasse:
    - Palavra-chave `super` e um ponto (.)

Observação de engenharia de software 9.1

Os métodos de uma subclasse não acessam membros `private` diretamente de sua superclasse. Uma subclasse pode alterar o estado de variáveis de instância `private` da superclasse somente por meio de métodos não-`private` fornecidos na superclasse e herdados pela subclasse.

Observação de engenharia de software 9.2

Declarar variáveis de instância `private` ajuda os programadores a testar, depurar e a modificar sistemas corretamente. Se uma subclasse pudesse acessar variáveis de instância `private` da sua superclasse, classes que herdam dessa subclasse também poderiam acessar as variáveis de instância. Isso propagaria acesso ao que devem ser variáveis de instância `private` e os benefícios do ocultamento de informações seriam perdidos.

9.4 Relacionamento entre superclasses e subclasses

- **Relacionamento entre superclasse e subclasse.**
  - **Exemplo: Hierarquia de herança `CommissionEmployee/BasePlusCommissionEmployee`**
    - `CommissionEmployee`
      - **Primeiro nome, sobrenome, SSN, taxa de comissão, total de vendas brutas.**
    - `BasePlusCommissionEmployee`
      - **Primeiro nome, sobrenome, SSN, taxa de comissão, total de vendas brutas.**
      - **Salário-base.**

9.4.1 Criando e utilizando uma classe `CommissionEmployee`

- Classe `CommissionEmployee`
  - Estende a classe `Object`.
    - Palavra-chave `extends`.
    - Toda classe no Java estende uma classe existente.
      - Exceto `Object`.
    - Toda classe herda os métodos de `Object`.
    - Uma nova classe estende implicitamente `Object`:
      - Se não estender uma outra classe.

Observação de engenharia de software 9.3

O compilador Java configura a superclasse de uma classe como `Object` quando a declaração de classe não estender uma superclasse explicitamente.

Resumo

nEmployee .java

de 4)

nha 4

Linhas 6-10

Linhas 20-21

```
1  // Fig. 9.4: CommissionEmployee.java
2  // Classe CommissionEmployee representa um empregado comi
3
4  public class CommissionEmployee extends Object
5  {
6     private String firstName;
7     private String lastName;
8     private String socialSecurityNumber;
9     private double grossSales; // vendas brutas semanais
10    private double commissionRate; // porcentagem da com
11
12    // construtor de cinco argumentos
13    public CommissionEmployee( String first, String last, String ssn,
14       double sales, double rate )
15    {
16       // chamada implícita para o construtor Object ocorre aqui
17       firstName = first;
18       lastName = last;
19       socialSecurityNumber = ssn;
20       setGrossSales( sales ); // valida e armazena vendas brutas
21       setCommissionRate( rate ); // valida e armazena a taxa de comissão
22    } // fim do construtor CommissionEmployee de cinco argumentos
23
24    // configura o nome
25    public void setFirstName( String first )
26    {
27       firstName = first;
28    } // fim do método setFirstName
29
```

Declara variáveis de instância `private`

A classe `CommissionEmployee` estende a classe `Object`

Chamada implícita ao construtor de `Object`

Inicializa variáveis de instância

Invoca os métodos `setGrossSales` e `setCommissionRate` para validar os dados

```
30    // retorna o nome
31    public String getFirstName()
32    {
33       return firstName;
34    } // fim do método getFirstName
35
36    // configura o último nome
37    public void setLastName( String last )
38    {
39       lastName = last;
40    } // fim do método setLastName
41
42    // retorna o sobrenome
43    public String getLastName()
44    {
45       return lastName;
46    } // fim do método getLastName
47
48    // configura o CIC
49    public void setSocialSecurityNumber( String ssn )
50    {
51       socialSecurityNumber = ssn; // deve validar
52    } // fim do método setSocialSecurityNumber
53
54    // retorna número do CIC
55    public String getSocialSecurityNumber()
56    {
57       return socialSecurityNumber;
58    } // fim do método getSocialSecurityNumber
59
```

Resumo

CommissionEmployee.java

(2 de 4)

Resumo

CommissionEmployee.java

(3 de 4)

Linhas 85-88

```
60    // configura a quantidade de vendas brutas
61    public void setGrossSales( double sales )
62    {
63       grossSales = ( sales < 0.0 ) ? 0.0 : sales;
64    } // fim do método setGrossSales
65
66    // retorna a quantidade de vendas brutas
67    public double getGrossSales()
68    {
69       return grossSales;
70    } // fim do método getGrossSales
71
72    // configura taxa de comissão
73    public void setCommissionRate( double rate )
74    {
75       commissionRate = ( rate > 0.0 && rate < 1.0 ) ? rate : 0.0;
76    } // fim do método setCommissionRate
77
78    // retorna taxa de commissão
79    public double getCommissionRate()
80    {
81       return commissionRate;
82    } // fim do método getCommissionRate
83
84    // calcula os lucros
85    public double earnings()
86    {
87       return commissionRate * grossSales;
88    } // fim do método earnings
89
```

Calcula earnings

```
90    // retorna a representação String do objeto CommissionEmployee
91    public String toString()
92    {
93       return String.format( "%s: %s %s\n%s: %s\n%s: %.2f\n%s: %.2f",
94          "commission employee", firstName, lastName,
95          "social security number", socialSecurityNumber,
96          "gross sales", grossSales,
97          "commission rate", commissionRate );
98    } // fim do método toString
99 } // fim da classe CommissionEmployee
```

Resumo

CommissionEmployee.java

(4 de 4)

Linhas 91-98

Erro comum de programação 9.1

É um erro de sintaxe sobrescrever um método com um modificador de acesso mais restrito — um método `public` da superclasse não pode se tornar um método `protected` ou `private` na subclasse; um método `protected` da superclasse não pode se tornar um método `private` na subclasse. Fazer isso quebraria o relacionamento 'é um' em que se exige que todos os objetos de subclasse sejam capazes de responder a chamadas de método que são feitas para os métodos `public` declarados na superclasse. (C*ontinua*...)

Erro comum de programação 9.1 *(Continuação)*

Se um método `public` pudesse ser sobrescrito como um método `protected` ou `private`, os objetos de subclasse não seriam capazes de responder às mesmas chamadas de método como objetos de superclasse. Uma vez que um método é declarado `public` em uma superclasse, o método permanece `public` para todas as subclasses diretas e indiretas dessa classe.

```
1  // Fig. 9.5: CommissionEmployeeTest.java
2  // Testando a classe CommissionEmployee.
3
4  public class CommissionEmployeeTest
5  {
6     public static void main( String args[] )
7     {
8        // instancia o objeto CommissionEmployee
9        CommissionEmployee employee = new CommissionEmployee(
10          "Sue", "Jones", "222-22-2222", 10000, .06 );
11
12       // obtém os dados de empregado comissionado
13       System.out.println(
14          "Employee information obtained by get methods: \n" );
15       System.out.printf( "%s %s\n", "First name is",
16          employee.getFirstName() );
17       System.out.printf( "%s %s\n", "Last name is",
18          employee.getLastName() );
19       System.out.printf( "%s %s\n", "Social security number is",
20          employee.getSocialSecurityNumber() );
21       System.out.printf( "%s %.2f\n", "Gross sales is",
22          employee.getGrossSales() );
23       System.out.printf( "%s %.2f\n", "Commission rate is",
24          employee.getCommissionRate() );
25
26       employee.setGrossSales( 500 ); // configura vendas brutas
27       employee.setCommissionRate( .1 ); // configura taxa de comissão
28
```

Resumo

CommissionEmployeeTest.java

(1 de 2)

Linhas 9-10

Linhas 15-25

Linha 26-27

Resumo

```
29       System.out.printf( "\n%s:\n\n%s\n",
30          "Updated employee information obtained by toString", employee );
31    } // fim de main
32 } // fim da classe CommissionEmployeeTest
```

```
Employee information obtained by get methods:

First name is Sue
Last name is Jones
Social security number is 222-22-2222
Gross sales is 10000.00
Commission rate is 0.06

Updated employee information obtained by toString:

commission employee: Sue Jones
social security number: 222-22-2222
gross sales: 500.00
commission rate: 0.10
```

Chama implicitamente o método `toString` do objeto

onEmployee
va

(2 de 2)

Linha 30

Saída do programa

9.4.2 Criando uma classe `BasePlusCommissionEmployee` sem utilizar herança

- Classe `BasePlusCommissionEmployee`
  - Estende implicitamente `Object`.
  - Boa parte do código é semelhante a `CommissionEmployee`:
    - variáveis de instância `private`;
    - métodos `public`; e
    - construtor.
  - Adições:
    - variável de instância `private` `baseSalary`; e
    - métodos `setBaseSalary` e `getBaseSalary`.

Resumo

BasePlusCommissionEmployee.java

```
1  // Fig. 9.6: BasePlusCommissionEmployee.java
2  // A classe BasePlusCommissionEmployee representa um empregado que recebe
3  // um salário-base além da comissão.
4
5  public class BasePlusCommissionEmployee
6  {
7     private String firstName;
8     private String lastName;
9     private String socialSecurityNumber;
10    private double grossSales; // vendas brutas semanais
11    private double commissionRate; // porcentagem da comissão
12    private double baseSalary; // salário-base por semana
13
14    // construtor de seis argumentos
15    public BasePlusCommissionEmployee( String first, String last,
16       String ssn, double sales, double rate, double salary )
17    {
18       // chamada implícita para o construtor Object ocorre aqui
19       firstName = first;
20       lastName = last;
21       socialSecurityNumber = ssn;
22       setGrossSales( sales ); // valida e armazena venda
23       setCommissionRate( rate ); // valida e armazena a taxa de comissão
24       setBaseSalary( salary ); // valida e armazena salário-base
25    } // fim do construtor BasePlusCommissionEmployee de seis argumentos
26
```

Adiciona a variável de instância `baseSalary`

Linha 12

Linha 24

Utiliza o método `setBaseSalary` para validar os dados

Resumo

BasePlusCommissionEmployee.java

(2 de 4)

```
27    // configura o nome
28    public void setFirstName( String first )
29    {
30       firstName = first;
31    } // fim do método setFirstName
32
33    // retorna o nome
34    public String getFirstName()
35    {
36       return firstName;
37    } // fim do método getFirstName
38
39    // configura o sobrenome
40    public void setLastName( String last )
41    {
42       lastName = last;
43    } // fim do método setLastName
44
45    // retorna o sobrenome
46    public String getLastName()
47    {
48       return lastName;
49    } // fim do método getLastName
50
51    // configura o CIC
52    public void setSocialSecurityNumber( String ssn )
53    {
54       socialSecurityNumber = ssn; // deve validar
55    } // fim do método setSocialSecurityNumber
56
```

Resumo

BasePlusCommissionEmployee.java

(3 de 4)

```
57    // configura o CIC
58    public String getSocialSecurityNumber()
59    {
60       return socialSecurityNumber;
61    } // fim do método getSocialSecurityNumber
62
63    // configura quantidade de vendas brutas
64    public void setGrossSales( double sales )
65    {
66       grossSales = ( sales < 0.0 ) ? 0.0 : sales;
67    } // fim do método setGrossSales
68
69    // retorna a quantidade de vendas brutas
70    public double getGrossSales()
71    {
72       return grossSales;
73    } // fim do método getGrossSales
74
75    // configura a taxa de comissão
76    public void setCommissionRate( double rate )
77    {
78       commissionRate = ( rate > 0.0 && rate < 1.0 ) ? rate : 0.0;
79    } // fim do método setCommissionRate
80
81    // retorna taxa de comissão
82    public double getCommissionRate()
83    {
84       return commissionRate;
85    } // fim do método getCommissionRate
86
```

Resumo

```
87    // configura salário-base
88    public void setBaseSalary( double salary )
89    {
90       baseSalary = ( salary < 0.0 ) ? 0.0 : salary;
91    } // fim do método setBaseSalary
92
93    // retorna salário-base
94    public double getBaseSalary()
95    {
96       return baseSalary;
97    } // fim do método getBaseSalary
98
99    // calcula lucros
100     public double earnings()
101     {
102        return baseSalary + ( commissionRate * grossSales );
103     } // fim do método earnings
104
105     // retorna a representação de String de BasePlusCommissionEmployee
106     public String toString()
107     {
108        return String.format(
109           "%s: %s %s\n%s: %s\n%s: %.2f\n%s: %.2f\n%s: %.2f",
110           "base-salaried commission employee", firstName, lastName,
111           "social security number", socialSecurityNumber,
112           "gross sales", grossSales, "commission rate", commissionRate,
113           "base salary", baseSalary );
114     } // fim do método toString
115  } // fim da classe BasePlusCommissionEmployee
```

O método `setBaseSalary` valida os dados e configura a variável de instância `baseSalary`

`PlusCommission`
`oyee.java`

(4 de 4)

Linhas 88-91

Linhas 94-97

Linha 102

Linhas 108-113

Atualiza o método `earnings` a fim de calcular os rendimentos da comissão-base de um empregado assalariado

Resumo

sePlusCommission ployeeTest.java

(1 de 2)

Linha 9-11

Linhas 16-27

Instancia o objeto `BasePlusCommissionEmployee`

```
1  // Fig. 9.7: BasePlusCommissionEmployeeTest.java
2  // Testando a classe BasePlusCommissionEmployee.
3
4  public class BasePlusCommissionEmployeeTest
5  {
6     public static void main( String args[] )
7     {
8        // instancia o objeto BasePlusCommissionEmployee
9        BasePlusCommissionEmployee employee =
10          new BasePlusCommissionEmployee(
11          "Bob", "Lewis", "333-33-3333", 5000, .04, 300 );
12
13       // obtém os dados do empregado comissionado com salário-base
14       System.out.println(
15          "Employee information obtained by get methods: \n" );
16       System.out.printf( "%s %s\n", "First name is",
17          employee.getFirstName() );
18       System.out.printf( "%s %s\n", "Last name is",
19          employee.getLastName() );
20       System.out.printf( "%s %s\n", "Social security number is",
21          employee.getSocialSecurityNumber() );
22       System.out.printf( "%s %.2f\n", "Gross sales is",
23          employee.getGrossSales() );
24       System.out.printf( "%s %.2f\n", "Commission rate is",
25          employee.getCommissionRate() );
26       System.out.printf( "%s %.2f\n", "Base salary is",
27          employee.getBaseSalary() );
28
```

Resumo

```
29       employee.setBaseSalary( 1000 ); // configura o salário-base
30
31       System.out.printf( "\n%s:\n\n%s\n",
32          "Updated employee information obtained by
33           employee.toString() );
34    } // fim de main
35 } // fim da classe BasePlusCommissionE
```

Utiliza os métodos `setBaseSalary` de `BasePlusCommissionEmployee` para configurar o salário-base

Chama explicitamente o método `toString` do objeto

```
Employee information obtained by get met

First name is Bob
Last name is Lewis
Social security number is 333-33-3333
Gross sales is 5000.00
Commission rate is 0.04
Base salary is 300.00

Updated employee information obtained by toString:

base-salaried commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: 5000.00
commission rate: 0.04
base salary: 1000.00
```

ssion
EmployeeTest.java

(2 de 2)

Linha 29

Linha 33

Saída do programa

Observação de engenharia de software 9.4

Copiar e colar código de uma classe para a outra pode espalhar erros por múltiplos arquivos de código-fonte. Para evitar a duplicação de código (e possivelmente erros), utilize herança, em vez da abordagem ‘copiar e colar’, em situações em que você quer que uma classe ‘absorva’ as variáveis de instância e métodos de outra classe.

Observação de engenharia de software 9.5

Com a herança, as variáveis de instância comuns e os métodos de todas as classes na hierarquia são declarados em uma superclasse. Quando as alterações são requeridas para esses recursos comuns, os desenvolvedores de software só precisam fazer as alterações na superclasse — as subclasses então herdam as alterações. Sem a herança, as alterações precisariam ser feitas em todos os arquivos de código-fonte que contêm uma cópia do código em questão.

9.4.3 Criando uma hierarquia de herança para CommissionEmployee-BasePlusCommissionEmployee

- Classe BasePlusCommissionEmployee2:
  - estende a classe CommissionEmployee;
  - é uma CommissionEmployee;
  - tem a variável de instância baseSalary;
  - herda membros public e protected; e
  - construtor não-herdado.

Resumo

BasePlusCommissionEmployee2.java

(1 de 3)

Linha 4

Linha 13

```
1  // Fig. 9.8: BasePlusCommissionEmployee2.java
2  // BasePlusCommissionEmployee2 herda da classe CommissionEmployee.
3
4  public class BasePlusCommissionEmployee2 extends CommissionEmployee
5  {
6     private double baseSalary; // salário-base por semana
7
8     // construtor de seis argumentos
9     public BasePlusCommissionEmployee2( String first, String last,
10       String ssn, double sales, double rate, double salary )
11    {
12       // chamada explícita para o construtor CommissionEmployee da superclasse
13       super( first, last, ssn, sales, rate );
14
15       setBaseSalary( amount ); // valida e armazena salário-base
16    } // fim do construtor BasePlusCommissionEmployee2 de seis argumentos
17
18    // configura salário-base
19    public void setBaseSalary( double salary )
20    {
21       baseSalary = ( salary < 0.0 ) ? 0.0 : salary;
22    } // fim do método setBaseSalary
23
```

Resumo

BasePlusCommission

(2 de 3)

Linha 34

Linhas 41-46

```
24    // retorna o salário-base
25    public double getBaseSalary()
26    {
27       return baseSalary;
28    } // fim do método getBaseSalary
29
30    // calcula os lucros
31    public double earnings()
32    {
33       // não permitido: commissionRate e grossSales private em superclasse
34       return baseSalary + ( commissionRate * grossSales );
35    } // fim do método earnings
36
37    // retorna representação
38    public String toString()
39    {
40       // não permitido: tent
41       return String.format(
42          "%s: %s %s\n%s: %s\n%s: %.2f\n%s: %.2f\n%s: %.2f",
43          "base-salaried commission employee", firstName, lastName,
44          "social security number", socialSecurityNumber,
45          "gross sales", grossSales, "commission rate", commissionRate,
46          "base salary", baseSalary );
47    } // fim do método toString
48 } // fim da classe BasePlusCommissionEmployee2
```

O compilador gera erros porque as variáveis de instância `commissionRate` e `grossSales` da superclasse são `private`

O compilador gera erros porque as variáveis de instância `firstName`, `lastName`, `socialSecurityNumber`, `grossSales` e `commissionRate` da superclasse são `private`

```
BasePlusCommissionEmployee2.java:34: commissionRate has private access in
CommissionEmployee
      return baseSalary + ( commissionRate * grossSales );
                            ^
BasePlusCommissionEmployee2.java:34: grossSales has private access in
CommissionEmployee
      return baseSalary + ( commissionRate * grossSales );
                                             ^
BasePlusCommissionEmployee2.java:43: firstName has private access in
CommissionEmployee
         "base-salaried commission employee", firstName, lastName,
                                              ^
BasePlusCommissionEmployee2.java:43: lastName has private access in
CommissionEmployee
         "base-salaried commission employee", firstName, lastName,
                                                         ^
BasePlusCommissionEmployee2.java:44: socialSecurityNumber has private access in
CommissionEmployee
         "social security number", socialSecurityNumber,
                                   ^
BasePlusCommissionEmployee2.java:45: grossSales has private access in
CommissionEmployee
         "gross sales", grossSales, "commission rate", commissionRate,
                        ^
BasePlusCommissionEmployee2.java:45: commissionRate has private access in
CommissionEmployee
         "gross sales", grossSales, "commission rate", commissionRate,
                                                      ^
7 errors
```

Resumo

BasePlusCommissionEmployee2.java

(3 de 3)

O compilador gerou erros

Erro comum de programação 9.2

Um erro de compilação ocorre se um construtor de subclasse chamar um de seus construtores de superclasse com argumentos que não correspondem exatamente ao número e tipos de parâmetros especificados em uma das declarações de construtor de superclasse.

9.4.4 Hierarquia de herança de `CommissionEmployee-BasePlusCommissionEmployee` com variáveis de instância protected (*Continuação*)

- **Utiliza variáveis de instância `protected`:**
  - **Permite que a classe `BasePlusCommissionEmployee` acesse diretamente as variáveis de instância da superclasse.**
  - **Os membros `protected` da superclasse são herdados por todas as subclasses dessa superclasse.**

Resumo

CommissionEmployee2.java

(1 de 4)

Linha 6-10

```
1  // Fig. 9.9: CommissionEmployee2.java
2  // Classe CommissionEmployee2 representa um empregado comissionado.
3
4  public class CommissionEmployee2
5  {
6     protected String firstName;
7     protected String lastName;
8     protected String socialSecurityNumber;
9     protected double grossSales; // vendas brutas semanais
10    protected double commissionRate; // porcentagem da comissão
11
12    // construtor de cinco argumentos
13    public CommissionEmployee2( String first, String last, String ssn,
14       double sales, double rate )
15    {
16       // chamada implícita para o construtor Object ocorre aqui
17       firstName = first;
18       lastName = last;
19       socialSecurityNumber = ssn;
20       setGrossSales( sales ); // valida e armazena as vendas brutas
21       setCommissionRate( rate ); // valida e armazena a taxa de comissão
22    } // fim do construtor CommissionEmployee2 de cinco argumentos
23
24    // configura o nome
25    public void setFirstName( String first )
26    {
27       firstName = first;
28    } // fim do método setFirstName
29
```

Declara variáveis de instância `protected`

```
30    // retorna o nome
31    public String getFirstName()
32    {
33       return firstName;
34    } // fim do método getFirstName
35
36    // configura o sobrenome
37    public void setLastName( String last )
38    {
39       lastName = last;
40    } // fim do método setLastName
41
42    // retorna o sobrenome
43    public String getLastName()
44    {
45       return lastName;
46    } // fim do método getLastName
47
48    // configura o CIC
49    public void setSocialSecurityNumber( String ssn )
50    {
51       socialSecurityNumber = ssn; // deve validar
52    } // fim do método setSocialSecurityNumber
53
54    // retorna CIC
55    public String getSocialSecurityNumber()
56    {
57       return socialSecurityNumber;
58    } // end method getSocialSecurityNumber
59
```

Resumo

Commission

`Employee2.java`

(2 de 4)

```
60    // configura a quantidade de vendas brutas
61    public void setGrossSales( double sales )
62    {
63       grossSales = ( sales < 0.0 ) ? 0.0 : sales;
64    } // fim do método setGrossSales
65
66    // retorna a quantidade de vendas brutas
67    public double getGrossSales()
68    {
69       return grossSales;
70    } // fim do método getGrossSales
71
72    // configura a taxa de comissão
73    public void setCommissionRate( double rate )
74    {
75       commissionRate = ( rate > 0.0 && rate < 1.0 ) ? rate : 0.0;
76    } // fim do método setCommissionRate
77
78    // retorna a taxa de comissão
79    public double getCommissionRate()
80    {
81       return commissionRate;
82    } // fim do método getCommissionRate
83
84    // calcula os lucros
85    public double earnings()
86    {
87       return commissionRate * grossSales;
88    } // fim do método earnings
89
```

Resumo

Commission

Employee2.java

(3 de 4)

```
90    // retorna a representação String do objeto CommissionEmployee2
91    public String toString()
92    {
93       return String.format( "%s: %s %s\n%s: %s\n%s: %.2f\n%s: %.2f",
94          "commission employee", firstName, lastName,
95          "social security number", socialSecurityNumber,
96          "gross sales", grossSales,
97          "commission rate", commissionRate );
98    } // fim do método toString
99 } // fim da classe CommissionEmployee2
```

Resumo

CommissionEmployee2.java

(4 de 4)

Resumo

BasePlusCommissionEmployee3.java

```
1  // Fig. 9.10: BasePlusCommissionEmployee3.java
2  // BasePlusCommissionEmployee3 herda de CommissionEmployee2 e tem
3  // acesso a membros protected de CommissionEmployee2.
4
5  public class BasePlusCommissionEmployee3 extends CommissionEmployee2
6  {
7     private double baseSalary; // salário-base por semana
8
9     // construtor de seis argumentos
10    public BasePlusCommissionEmployee3( String first, String
11       String ssn, double sales, double rate, double salary )
12    {
13       super( first, last, ssn, sales, rate );
14       setBaseSalary( salary ); // valida e armazena salário-base
15    } // fim do construtor BasePlusCommissionEmployee3 de seis argumentos
16
17    // configura salário-base
18    public void setBaseSalary( double salary )
19    {
20       baseSalary = ( salary < 0.0 ) ? 0.0 : salary;
21    } // fim do método setBaseSalary
22
23    // retorna salário-base
24    public double getBaseSalary()
25    {
26       return baseSalary;
27    } // fim do método getBaseSalary
28
```

Deve chamar o construtor da superclasse

Linha 13

Resumo

```
29    // calcula os lucros
30    public double earnings()
31    {
32       return baseSalary + ( commissionRate * grossSales );
33    } // fim do método earnings
34
35    // retorna a representação String de BasePlusCommissionEmployee3
36    public String toString()
37    {
38       return String.format(
39          "%s: %s %s\n%s: %s\n%s: %.2f\n%s: %.2f\n%s: %.2f",
40          "base-salaried commission employee", firstName, lastName,
41          "social security number", socialSecurityNumber,
42          "gross sales", grossSales, "commission rate", commissionRate,
43          "base salary", baseSalary );
44    } // fim do método toString
45 } // fim da classe BasePlusCommissionEmployee3
```

Acessa diretamente as variáveis de instância `protected` da superclasse

BasePlusCommissionEmployee3.java

Linha 32

Linhas 38-43

Resumo

BasePlusCommissionEmployeeTest3.java

(1 de 2)

```
1  // Fig. 9.11: BasePlusCommissionEmployeeTest3.java
2  // Testando a classe BasePlusCommissionEmployee3.
3
4  public class BasePlusCommissionEmployeeTest3
5  {
6     public static void main( String args[] )
7     {
8        // instancia o objeto BasePlusCommissionEmployee3
9        BasePlusCommissionEmployee3 employee =
10          new BasePlusCommissionEmployee3(
11          "Bob", "Lewis", "333-33-3333", 5000, .04, 300 );
12
13       // obtém os dados do empregado comissionado com salário-base
14       System.out.println(
15          "Employee information obtained by get methods: \n" );
16       System.out.printf( "%s %s\n", "First name is",
17          employee.getFirstName() );
18       System.out.printf( "%s %s\n", "Last name is",
19          employee.getLastName() );
20       System.out.printf( "%s %s\n", "Social security number is",
21          employee.getSocialSecurityNumber() );
22       System.out.printf( "%s %.2f\n", "Gross sales is",
23          employee.getGrossSales() );
24       System.out.printf( "%s %.2f\n", "Commission rate is",
25          employee.getCommissionRate() );
26       System.out.printf( "%s %.2f\n", "Base salary is",
27          employee.getBaseSalary() );
28
```

```
29       employee.setBaseSalary( 1000 ); // configura o salário-base
30
31       System.out.printf( "\n%s:\n\n%s\n",
32          "Updated employee information obtained by toString",
33          employee.toString() );
34    } // fim de main
35 } // fim da classe BasePlusCommissionEmployeeTest3
```

```
Employee information obtained by get methods:

First name is Bob
Last name is Lewis
Social security number is 333-33-3333
Gross sales is 5000.00
Commission rate is 0.04
Base salary is 300.00

Updated employee information obtained by toString:

base-salaried commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: 5000.00
commission rate: 0.04
base salary: 1000.00
```

Resumo

BasePlusCommissionEmployeeTest3.java

(2 de 2)

Saída do programa

9.4.4 Hierarquia de herança de `CommissionEmployee-BasePlusCommissionEmployee` utilizando variáveis de instância protegidas (*Continuação*)

- **Utilizando variáveis de instância `protected`.**
  - **Vantagens:**
    - **subclasses podem modificar valores diretamente; e**
    - **pequeno aumento no desempenho.**
      - **Evita overheads da chamada aos métodos *set*/*get*.**
  - **Desvantagens:**
    - **Nenhum teste de validade:**
      - **subclasse pode atribuir valor ilegal.**
    - **Dependente da implementação:**
      - **métodos da subclasse mais provavelmente dependentes da implementação da superclasse; e**
      - **alterações na implementação da superclasse podem resultar em modificações da subclasse.**
        - **Software frágil (quebradiço).**

Observação de engenharia de software 9.6

Utilize o modificador de acesso `protected` quando uma superclasse precisar fornecer um método somente para suas subclasses e outras classes no mesmo pacote, mas não para outros clientes.

Observação de engenharia de software 9.7

Declarar as variáveis de instância da superclasse `private` (em oposição a `protected`) permite a implementação de superclasse dessas variáveis de instância para alterar sem afetar as implementações de subclasse.

Dica de prevenção de erro 9.1

Quando possível, não inclua variáveis de instância `protected` em uma superclasse. Em vez disso, inclua métodos não-`private` que acessam as variáveis de instância `private`. Isso irá assegurar que os objetos da classe mantenham estados consistentes.

9.4.5 Hierarquia de herança de `CommissionEmployee–BasePlusCommissionEmployee` com variáveis de instância `private`

- **Reexamine nossa hierarquia de novo:**
  - **Agora utilizando práticas de engenharia de software melhores.**
    - **Declare as variáveis de instância como `private`.**
    - **Forneça os métodos *get* e *set* públicos.**
    - **Utilize o método *get* para obter os valores das variáveis de instância.**

Resumo

CommissionEmployee3.java

(1 de 4)

Linhas 6-10

```
1  // Fig. 9.12: CommissionEmployee3.java
2  // A classe CommissionEmployee3 representa um empregado comissionado.
3
4  public class CommissionEmployee3
5  {
6     private String firstName;
7     private String lastName;
8     private String socialSecurityNumber;
9     private double grossSales; // vendas brutas semanais
10    private double commissionRate; // porcentagem da comissão
11
12    // construtor de cinco argumentos
13    public CommissionEmployee3( String first, String last, String ssn,
14       double sales, double rate )
15    {
16       // chamada implícita para o construtor Object ocorre aqui
17       firstName = first;
18       lastName = last;
19       socialSecurityNumber = ssn;
20       setGrossSales( sales ); // valida e armazena as vendas brutas
21       setCommissionRate( rate ); // valida e armazena a taxa de comissão
22    } // fim do construtor CommissionEmployee3 de cinco argumentos
23
24    // configura o nome
25    public void setFirstName( String first )
26    {
27       firstName = first;
28    } // fim do método setFirstName
29
```

Declara variáveis de instância `private`

```
30    // configura o nome
31    public String getFirstName()
32    {
33       return firstName;
34    } // fim do método getFirstName
35
36    // configura o sobrenome
37    public void setLastName( String last )
38    {
39       lastName = last;
40    } // fim do método setLastName
41
42    // retorna o sobrenome
43    public String getLastName()
44    {
45       return lastName;
46    } // fim do método getLastName
47
48    // configura o CIC
49    public void setSocialSecurityNumber( String ssn )
50    {
51       socialSecurityNumber = ssn; // deve validar
52    } // fim do método setSocialSecurityNumber
53
54    // retorna o CIC
55    public String getSocialSecurityNumber()
56    {
57       return socialSecurityNumber;
58    } // fim do método getSocialSecurityNumber
59
```

Resumo

Commission

Employee3.java

(2 de 4)

```
60    // configura a quantidade de vendas brutas
61    public void setGrossSales( double sales )
62    {
63       grossSales = ( sales < 0.0 ) ? 0.0 : sales;
64    } // fim do método setGrossSales
65
66    // retorna a quantidade de vendas brutas
67    public double getGrossSales()
68    {
69       return grossSales;
70    } // fim do método getGrossSales
71
72    // configura a taxa de comissão
73    public void setCommissionRate( double rate )
74    {
75       commissionRate = ( rate > 0.0 && rate < 1.0 ) ? rate : 0.0;
76    } // fim do método setCommissionRate
77
78    // retorna a taxa de comissão
79    public double getCommissionRate()
80    {
81       return commissionRate;
82    } // fim do método getCommissionRate
83
```

Resumo

Commission

Employee3.java

(3 de 4)

Resumo

```
84    // calcula os lucros
85    public double earnings()
86    {
87       return getCommissionRate() * getGrossSales();
88    } // fim do método earnings
89
90    // retorna a representação String do objeto CommissionEmpl
91    public String toString()
92    {
93       return String.format( "%s: %s %s\n%s: %s\n%s: %.2f\n%s: %.2f",
94          "commission employee", getFirstName(), getLastName(),
95          "social security number", getSocialSecurityNumber(),
96          "gross sales", getGrossSales(),
97          "commission rate", getCommissionRate() );
98    } // fim do método toString
99 } // fim da classe CommissionEmployee3
```

Utiliza os métodos *get* para obter os valores das variáveis de instância

Employee3.java

(4 de 4)

Linha 87

Linhas 94-97

Resumo

BasePlusCommissionEmployee4.java

(1 de 2)

```
1  // Fig. 9.13: BasePlusCommissionEmployee4.java
2  // Classe BasePlusCommissionEmployee4 herda de CommissionEmployee3 e
3  // acessa os dados privados de CommissionEmployee3 via os métodos
4  // CommissionEmployee3 e public.
5
6  public class BasePlusCommissionEmployee4 extends CommissionEmployee3
7  {
8     private double baseSalary; // salário-base por semana
9
10    // construtor de seis argumentos
11    public BasePlusCommissionEmployee4( String first, String last,
12       String ssn, double sales, double rate, double salary )
13    {
14       super( first, last, ssn, sales, rate );
15       setBaseSalary( salary ); // valida e armazena salário-base
16    } // fim do construtor BasePlusCommissionEmployee4 de seis argumentos
17
18    // configura o salário-base
19    public void setBaseSalary( double salary )
20    {
21       baseSalary = ( salary < 0.0 ) ? 0.0 : salary;
22    } // fim do método setBaseSalary
23
```

Herda de `CommissionEmployee3`

Resumo

```
24    // retorna o salário-base
25    public double getBaseSalary()
26    {
27       return baseSalary;
28    } // fim do método getBaseSalary
29
30    // calcula os lucros
31    public double earnings()
32    {
33       return getBaseSalary() + super.earnings();
34    } // fim do método earnings
35
36    // retorna a representação String de BasePlusCommissionEmplo
37    public String toString()
38    {
39       return String.format( "%s %s\n%s: %.2f", "base-salaried",
40          super.toString(), "base salary", getBaseSalary() );
41    } // fim do método toString
42 } // fim da classe BasePlusCommissionEmployee4
```

Invoca um método sobrescrito da superclasse a partir de uma subclasse

Utiliza os métodos *get* para obter os valores das variáveis de instância

Linhas 40

Invoca um método sobrescrito da superclasse a partir de uma subclasse

Erro comum de programação 9.3

Quando um método da superclasse é sobrescrito em uma subclasse, a versão da subclasse freqüentemente chama a versão da superclasse para realizar uma parte do trabalho. A falha em prefixar o nome do método da superclasse com a palavra-chave `super` e um ponto (.) separador ao referenciar o método da superclasse faz com que o método da subclasse chame a si mesmo, criando um erro chamado recursão infinita. A recursão, utilizada corretamente, é uma capacidade poderosa discutida no Capítulo 15, Recursão.

Resumo

```
1  // Fig. 9.14: BasePlusCommissionEmployeeTest4.java
2  // Testando a classe BasePlusCommissionEmployee4.
3
4  public class BasePlusCommissionEmployeeTest4
5  {
6     public static void main( String args[] )
7     {
8        // instancia o objeto BasePlusCommissionEmployee4
9        BasePlusCommissionEmployee4 employee =
10          new BasePlusCommissionEmployee4(
11          "Bob", "Lewis", "333-33-3333", 5000, .04, 300 );
12
13       // obtém os dados do empregado comissionado com salário-base
14       System.out.println(
15          "Employee information obtained by get methods: \n" );
16       System.out.printf( "%s %s\n", "First name is",
17          employee.getFirstName() );
18       System.out.printf( "%s %s\n", "Last name is",
19          employee.getLastName() );
20       System.out.printf( "%s %s\n", "Social security number is",
21          employee.getSocialSecurityNumber() );
22       System.out.printf( "%s %.2f\n", "Gross sales is",
23          employee.getGrossSales() );
24       System.out.printf( "%s %.2f\n", "Commission rate is",
25          employee.getCommissionRate() );
26       System.out.printf( "%s %.2f\n", "Base salary
27          employee.getBaseSalary() );
28
```

ion
ava

Cria um objeto `BasePlusCommissionEmployee4`.

(1 de 2)

Linhas 9-11

Linhas 16-25

Utiliza os métodos *get* herdados para acessar as variáveis de instância `private` herdadas

Utiliza o método *get* de `BasePlusCommissionEmployee4` para acessar a variável de instância privada.

```
29       employee.setBaseSalary( 1000 ); // configura o salário-base
30
31       System.out.printf( "\n%s:\n\n%s\n",
32          "Updated employee information obtaine
33          employee.toString() );
34    } // fim de main
35 } // fim da classe BasePlusCommissionEmployeeT
```

Utiliza o método *set* de `BasePlusCommissionEmployee4` para modificar a variável de instância `private baseSalary`.

```
Employee information obtained by get methods:

First name is Bob
Last name is Lewis
Social security number is 333-33-3333
Gross sales is 5000.00
Commission rate is 0.04
Base salary is 300.00

Updated employee information obtained by toString:

base-salaried commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: 5000.00
commission rate: 0.04
base salary: 1000.00
```

(2 de 2)

9.5 Construtores em subclasses

- **Instanciando objetos de subclasse.**
  - **Cadeia de chamadas ao construtor:**
    - **O construtor da subclasse invoca o construtor da superclasse implícita ou explicitamente.**
    - **Base da hierarquia de herança:**
      - **O último construtor chamado na cadeia é o construtor de `Object`.**
      - **O corpo do construtor de subclasse original termina a execução por último.**
      - **Exemplo: Hierarquia de `CommissionEmployee3-BasePlusCommissionEmployee4`**
        - **O construtor de `CommissionEmployee3` chamado em penúltimo lugar (o último é o construtor de `Object`).**
        - **O corpo do construtor de `CommissionEmployee3` termina a execução em segundo lugar (o primeiro é o corpo do construtor de `Object`).**

Observação de engenharia de software 9.8

Quando um programa cria um objeto de subclasse, o construtor de subclasse imediatamente chama o construtor de superclasse (explicitamente, via `super` ou implicitamente). O corpo do construtor de superclasse executa para inicializar as variáveis de instância da superclasse que fazem parte do objeto de subclasse, então o corpo do construtor de subclasse executa para inicializar variáveis de instância somente de subclasse. (C*ontinua*…)

Observação de engenharia de software 9.8 (*Continuação)*

O Java assegura que mesmo que um construtor não atribua um valor a uma variável de instância, a variável ainda é inicializada como seu valor-padrão (por exemplo, `0` para tipos numéricos primitivos, `false` para `boolean`s, `null` para referências).

Resumo

CommissionEmployee4.java

(1 de 4)

Linhas 23-24

```
1  // Fig. 9.15: CommissionEmployee4.java
2  // Classe CommissionEmployee4 representa um empregado comissionado.
3
4  public class CommissionEmployee4
5  {
6     private String firstName;
7     private String lastName;
8     private String socialSecurityNumber;
9     private double grossSales; // vendas brutas semanais
10    private double commissionRate; // porcentagem da comissão
11
12    // construtor de argumentos
13    public CommissionEmployee4( String first, String last, String ssn,
14       double sales, double rate )
15    {
16       // chamada implícita para o construtor Object ocorre aqui
17       firstName = first;
18       lastName = last;
19       socialSecurityNumber = ssn;
20       setGrossSales( sales ); // valida e ar
21       setCommissionRate( rate ); // valida e
22
23       System.out.printf(
24          "\nCommissionEmployee4 constructor:\n%s\n", this );
25    } // fim do construtor CommissionEmployee4 de cinco argumentos
26
```

O construtor gera uma mensagem para demonstrar a ordem de chamada de método.

Resumo

CommissionEmployee 4.java

(2 de 4)

```
27    // configura o nome
28    public void setFirstName( String first )
29    {
30       firstName = first;
31    } // fim do método setFirstName
32
33    // retorna o nome
34    public String getFirstName()
35    {
36       return firstName;
37    } // fim do método getFirstName
38
39    // configura o sobrenome
40    public void setLastName( String last )
41    {
42       lastName = last;
43    } // fim do método setLastName
44
45    // retorna o sobrenome
46    public String getLastName()
47    {
48       return lastName;
49    } // fim do método getLastName
50
51    // configura o CIC
52    public void setSocialSecurityNumber( String ssn )
53    {
54       socialSecurityNumber = ssn; // deve validar
55    } // fim do método setSocialSecurityNumber
56
```

```
57    // retorna o CIC
58    public String getSocialSecurityNumber()
59    {
60       return socialSecurityNumber;
61    } // fim do método getSocialSecurityNumber
62
63    // configura a quantidade de vendas brutas
64    public void setGrossSales( double sales )
65    {
66       grossSales = ( sales < 0.0 ) ? 0.0 : sales;
67    } // fim do método setGrossSales
68
69    // retorna a quantidade de vendas brutas
70    public double getGrossSales()
71    {
72       return grossSales;
73    } // fim do método getGrossSales
74
75    // configura a taxa de comissão
76    public void setCommissionRate( double rate )
77    {
78       commissionRate = ( rate > 0.0 && rate < 1.0 ) ? rate : 0.0;
79    } // fim do método setCommissionRate
80
```

Resumo

CommissionEmployee 4.java

(3 de 4)

```
81    // retorna a taxa de comissão
82    public double getCommissionRate()
83    {
84       return commissionRate;
85    } // fim do método getCommissionRate
86
87    // calcula os lucros
88    public double earnings()
89    {
90       return getCommissionRate() * getGrossSales();
91    } // fim do método earnings
92
93    // retorna a representação String de objeto CommissionEmployee4
94    public String toString()
95    {
96       return String.format( "%s: %s %s\n%s: %s\n%s: %.2f\n%s: %.2f",
97          "commission employee", getFirstName(), getLastName(),
98          "social security number", getSocialSecurityNumber(),
99          "gross sales", getGrossSales(),
100         "commission rate", getCommissionRate() );
101    } // fim do método toString
102 } // fim da classe CommissionEmployee4
```

Resumo

CommissionEmployee4.java

(4 de 4)

Resumo

BasePlusCommissionEmployee5.java

(1 de 2)

Linhas 15-16

```
1  // Fig. 9.16: BasePlusCommissionEmployee5.java
2  // Declaração de classe BasePlusCommissionEmployee5.
3
4  public class BasePlusCommissionEmployee5 extends CommissionEmployee4
5  {
6     private double baseSalary; // salário-base por semana
7
8     // construtor de seis argumentos
9     public BasePlusCommissionEmployee5( String first, String last,
10       String ssn, double sales, double rate, double salary )
11    {
12       super( first, last, ssn, sales, rate );
13       setBaseSalary( salary ); // valida e ar
14
15       System.out.printf(
16          "\nBasePlusCommissionEmployee5 constructor:\n%s\n", this );
17    } // fim do construtor BasePlusCommissionEmployee5 de seis argumentos
18
19    // configura salário-base
20    public void setBaseSalary( double salary )
21    {
22       baseSalary = ( salary < 0.0 ) ? 0.0 : salary;
23    } // fim do método setBaseSalary
24
```

O construtor gera uma mensagem para demonstrar a ordem de chamada de método.

```
25    // retorna salário-base
26    public double getBaseSalary()
27    {
28       return baseSalary;
29    } // fim do método getBaseSalary
30
31    // calcula os lucros
32    public double earnings()
33    {
34       return getBaseSalary() + super.earnings();
35    } // fim do método earnings
36
37    // retorna a representação String de BasePlusCommissionEmployee5
38    public String toString()
39    {
40       return String.format( "%s %s\n%s: %.2f", "base-salaried",
41          super.toString(), "base salary", getBaseSalary() );
42    } // fim do método toString
43 } // fim da classe BasePlusCommissionEmployee5
```

Resumo

BasePlusCommissionEmployee5.java

(2 de 2)

Resumo

```
1  // Fig. 9.17: ConstructorTest.java
2  // Exibe ordem em que construtores de superclasse e subclasse são chamados.
3
4  public class ConstructorTest
5  {
6     public static void main( String args[] )
7     {
8        CommissionEmployee4 employee1 = new CommissionEmployee4(
9           "Bob", "Lewis", "333-33-3333", 5000, .04 );
10
11       System.out.println();
12       BasePlusCommissionEmployee5 employee2 =
13          new BasePlusCommissionEmployee5(
14          "Lisa", "Jones", "555-55-5555", 2000, .06, 800 );
15
16       System.out.println();
17       BasePlusCommissionEmployee5 employee3 =
18          new BasePlusCommissionEmployee5(
19          "Mark", "Sands", "888-88-8888", 8000, .15, 2000 );
20    } // fim de main
21 } // fim da classe ConstructorTest
```

Instancia o objeto `CommissionEmployee4`

Instancia dois objetos `BasePlusCommissionEmployee5` para demonstrar a ordem das chamadas de método do construtor da subclasse e da superclasse.

torTest

.java

(1 de 2)

Resumo

ConstructorTest.java

(2 de 2)

```
CommissionEmployee4 constructor:
commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: 5000.00
commission rate: 0.04

CommissionEmployee4 constructor:
base-salaried commission employee: Lisa Jones
social security number: 555-55-5555
gross sales: 2000.00
commission rate: 0.06
base salary: 0.00

BasePlusCommissionEmployee5 constructor:
base-salaried commission employee: Lisa Jones
social security number: 555-55-5555
gross sales: 2000.00
commission rate: 0.06
base salary: 800.00


CommissionEmployee4 constructor:
base-salaried commission employee: Mark Sands
social security number: 888-88-8888
gross sales: 8000.00
commission rate: 0.15
base salary: 0.00

BasePlusCommissionEmployee5 constructor:
base-salaried commission employee: Mark Sands
social security number: 888-88-8888
gross sales: 8000.00
commission rate: 0.15
base salary: 2000.00
```

O corpo do construtor da subclasse `BasePlusCommissionEmployee5` é executado depois que a execução do construtor da superclasse `CommissionEmployee4` termina.

9.6 Engenharia de software com herança

- **Personalizando software existente:**
  - **Herda de classes existentes:**
    - **Inclui membros adicionais.**
    - **Redefine os membros da superclasse.**
    - **Nenhum acesso direto ao código-fonte da superclasse.**
      - **Vincula ao código-objeto.**
  - **Fornecedores de software independentes (*independent software vendors* – ISVs):**
    - **Desenvolve código proprietário para venda/licenciamento.**
      - **Disponível no formato de código-objeto.**
    - **Usuários derivam novas classes.**
      - **Sem acessar o código-fonte proprietário do ISV.**

Observação de engenharia de software 9.9

Apesar do fato de que herdar de uma classe não requer acesso ao código-fonte da classe, freqüentemente, os desenvolvedores insistem em examinar o código-fonte para entender como a classe é implementada. Os desenvolvedores na indústria querem assegurar que eles estão estendendo uma classe sólida — por exemplo, uma classe que executa bem e é implementada seguramente.

Observação de engenharia de software 9.10

Na etapa de projeto em um sistema orientado a objetos, o projetista freqüentemente descobre que certas classes estão intimamente relacionadas. O projetista deve ‘fatorar’ as variáveis de instância e métodos comuns e colocá-los em uma superclasse. Então o projetista deve utilizar a herança para desenvolver subclasses, especializando-as com capacidades além daquelas herdadas da superclasse.

Observação de engenharia de software 9.11

Declarar uma subclasse não afeta o código-fonte da sua superclasse. A herança preserva a integridade da superclasse.

Observação de engenharia de software 9.12

Assim como os projetistas de sistemas não-orientados a objetos devem evitar a proliferação de métodos, os projetistas de sistemas orientados a objetos devem evitar a proliferação de classes. Essa proliferação cria problemas de gerenciamento e pode prejudicar a capacidade de reutilização de software, porque em uma enorme biblioteca de classes torna-se difícil para um cliente localizar as classes mais apropriadas. A alternativa é criar menos classes que fornecem funcionalidades mais substanciais, mas isso pode se tornar complicado.

Dica de desempenho 9.1

Se as subclasses são maiores do que precisam ser (isto é, se elas contêm funcionalidades demais), recursos de memória e processamento podem ser desperdiçados. Estenda a superclasse que contém as funcionalidades mais próximas das funcionalidades que precisam ser criadas.

9.7 A classe `Object`

- **Métodos da classe `Object`:**
  - `clone`
  - `equals`
  - `finalize`
  - `getClass`
  - `hashCode`
  - `notify`, `notifyAll`, `wait`
  - `toString`

Método	Descrição
`Clone`	Esse método **`protected`**, que não aceita nenhum argumento e retorna uma referência **`Object`**, faz uma cópia do objeto em que é chamado. Quando a clonagem for necessária para os objetos de uma classe, a classe deve sobrescrever o método **`clone`** como um método **`public`** e deve implementar a interface **`Cloneable`** (pacote **`java.lang`**). A implementação padrão desse método realiza a chamada **cópia superficial** — os valores da variável de instância em um objeto são copiados em outro objeto do mesmo tipo. Para tipos por referência, apenas as referências são copiadas. Uma típica implementação do método **`clone`** sobrescrito realizaria uma **cópia em profundidade** que cria um novo objeto para cada variável de instância de tipo por referência. Há muitas sutilezas para sobrescrever método **`clone`**. Você pode aprender mais sobre a clonagem no seguinte artigo: **`java.sun.com/developer/JDCTechTips/2001/tt0306.html`**

Figura 9.18 | Métodos `Object` que são herdados direta ou indiretamente por todas as classes. (Parte 1 de 4.)

Método	Descrição
`Equals`	**Esse método compara dois objetos quanto à igualdade e retorna `true` se eles forem iguais, caso contrário, retorna `false`. O método aceita qualquer `Object` como um argumento. Quando os objetos de uma classe particular precisarem ser comparados quanto à igualdade, a classe deve sobrescrever o método `equals` para comparar o conteúdo dos dois objetos. A implementação do método deve atender aos seguintes requisitos:** • **Você deve retornar `false` se o argumento for `null`.** • **Você deve retornar `true` se um objeto for comparado com ele mesmo, como em `object1.equals( object1 )`.** • **Você só deve retornar `true` se tanto `object1.equals( object2 )` como `object2.equals( object1 )` retornarem `true`.** • **Para três objetos, se `object1.equals( object2 )` retornar `true` e `object2.equals( object3 )` retornar `true`, então `object1.equals( object3 )` também deve retornar `true`.** • **Se `equals` for chamado múltiplas vezes com os dois objetos e os objetos não mudarem, o método deve retornar `true` consistentemente se os objetos forem iguais e, `false`, caso contrário.** **Uma classe que sobrescreve `equals` também deve sobrescrever `hashCode` para assegurar que objetos iguais tenham códigos de hash idênticos. A implementação `equals` padrão utiliza o operador == para determinar se duas referências *referenciam o mesmo objeto* na memória. A Seção 29.3.3 demonstra o método `equals` da classe `String` e diferencia entre comparar objetos `String` com == e com `equals`.**

Figura. 9.18 | Métodos `Object` que são herdados direta ou indiretamente por todas as classes. (Parte 2 de 4.)

Método	Descrição
`finalize`	Esse método `protected` (introduzido na Seção 8.10 e Seção 8.11) é chamado pelo coletor de lixo para realizar a limpeza de término em um objeto antes de o coletor de lixo reivindicar a memória do objeto. Não é garantido que o coletor de lixo irá reivindicar um objeto, então não é possível garantir que o método `finalize` do objeto executará. O método deve especificar uma lista vazia de parâmetro e deve retornar `void`. A implementação padrão desse método serve como um marcador de lugar que não faz nada.
`getClass`	Todo objeto no Java conhece seu próprio tipo em tempo de execução. O método `getClass` (utilizado na Seção 10.5 e Seção 21.3) retorna um objeto de classe `Class` (pacote `java.lang`) que contém as informações sobre o tipo de objeto, como seu nome de classe (retornado pelo método `Class` `getName`). Você pode aprender mais sobre a classe `Class` na documentação de API on-line em `java.sun.com/j2se/5.0/docs/api/java/lang/Class.html`.

Figura 9.18 | Os métodos `Object` que são herdados direta ou indiretamente por todas as classes. (Parte 3 de 4.)

Método	Descrição
`hashCode`	Uma tabela de hash é uma estrutura de dados (discutida na Seção 19.10) que relaciona um objeto, chamado chave, com outro objeto, chamado valor. Ao inserir inicialmente um valor em uma tabela de hash, o método `hashCode` da chave é chamado. O valor do código de hash retornado é utilizado pela tabela de hash para determinar em qual localização inserir o valor correspondente. O código de hash da chave também é utilizado pela tabela de hash para localizar o valor correspondente da chave.
`notify`, `notifyAll`, `wait`	Os métodos `notify`, `notifyAll` e as três versões sobrecarregadas de `wait` são relacionados à multithreading, que é discutido no Capítulo 23. No J2SE 5.0, o modelo multithreading mudou substancialmente, mas esses recursos continuam a ser suportados.
`toString`	Esse método (introduzido na Seção 9.4.1) retorna uma representação `String` de um objeto. A implementação-padrão desse método retorna o nome de pacote e o nome de classe da classe do objeto seguido por uma representação hexadecimal do valor retornado pelo método `hashCode` do objeto.

Figura 9.18 | Métodos `Object` que são herdados direta ou indiretamente por todas as classes. (Parte 4 de 4.)