Java: como programar
Programação Orientada à Objetos

- Douglas Baptista de Godoy

/in/douglasbgodoy

github.com/douglasbgodoy

Fatec
Faculdade de Tecnologia

Etec
Escola Técnica Estadual

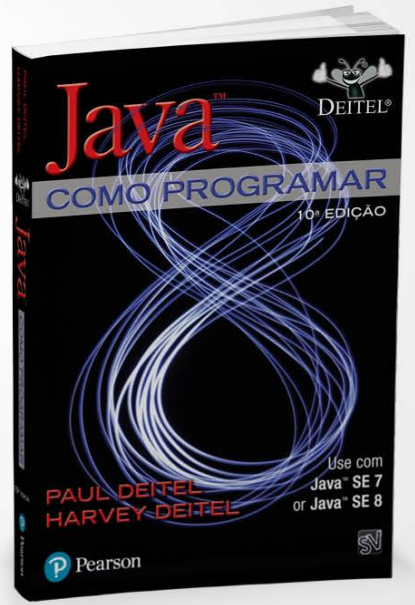
INSTITUTO FEDERAL
DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
São Paulo

Informação

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

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

Capítulo 9:
Programação orientada a objetos: herança



Introdução

- A **herança** reduz o tempo de desenvolvimento de programas.

- A **superclasse direta** de uma subclasse é aquela a partir da qual a subclasse é herdada.

- A **superclasse indireta** de uma subclasse está dois ou mais níveis acima da **hierarquia de classe** dessa subclasse.

Introdução

- Na **herança única**, uma classe deriva de uma superclasse.

- Na **herança múltipla**, uma classe é derivada de mais de uma superclasse direta.

O **Java não suporta herança múltipla**.

- Uma subclasse é mais específica que sua superclasse e representa um grupo menor de objetos.

- Cada objeto de uma subclasse também é um objeto da superclasse dessa classe.

- Entretanto, um objeto de superclasse não é um objeto de subclasses de sua classe.

Introdução

- Um **relacionamento *é um*** representa a herança.

- Em um relacionamento *é um*, um objeto de uma subclasse também pode ser tratado como um objeto de sua superclasse.

- Um **relacionamento *tem um*** representa composição.

- Em um relacionamento *tem um*, um objeto de classe contém referências a objetos de outras classes.

Superclasses e subclasses

- Os relacionamentos de herança simples formam estruturas hierárquicas do tipo árvore — há uma **superclasse** em um relacionamento hierárquico com suas **subclasses**.

Superclasse	Subclasses
`Student`	`GraduateStudent`, `UndergraduateStudent`
`Shape`	`Circle`, `Triangle`, `Rectangle`, `Sphere`, `Cube`
`Loan`	`CarLoan`, `HomeImprovementLoan`, `MortgageLoan`
`Employee`	`Faculty`, `Staff`
`BankAccount`	`CheckingAccount`, `SavingsAccount`

Figura 9.1 | Exemplos de herança.

Superclasses e subclasses

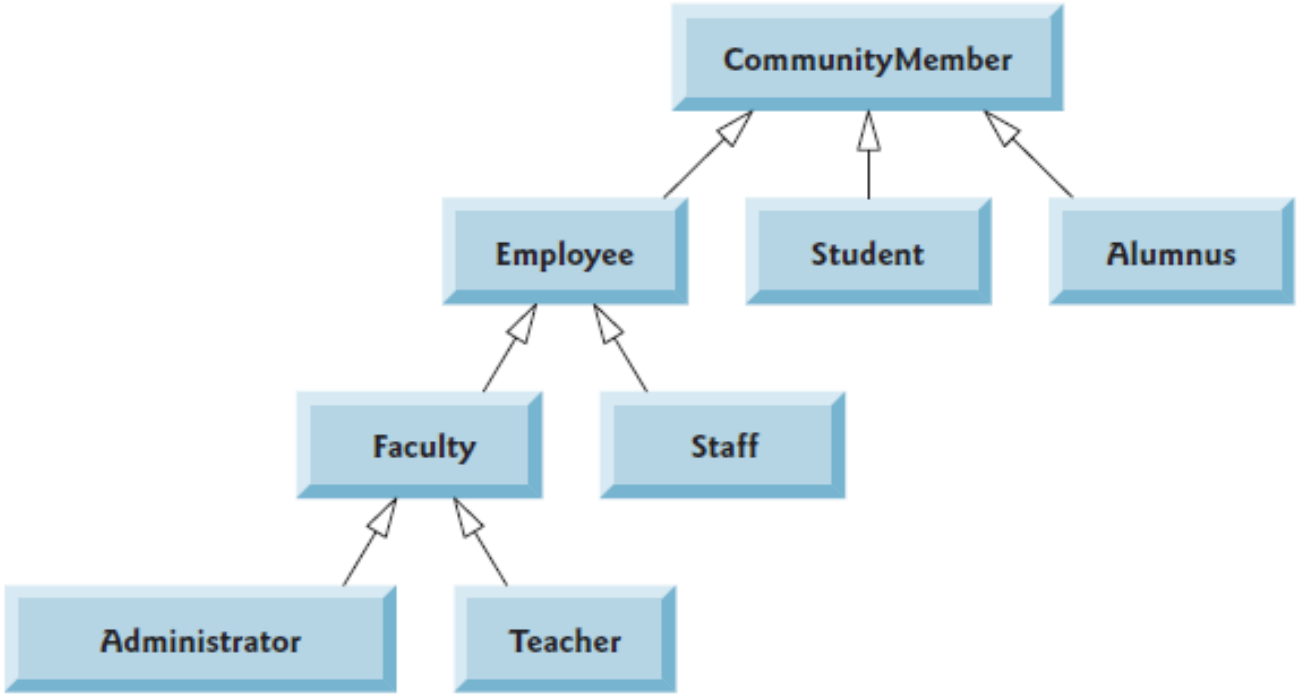


Figura 9.2 | Diagrama de classes UML da hierarquia de herança para `CommunityMembers` universitários.

Relacionamento entre superclasses e subclasses

- Uma subclasse não pode acessar os membros `private` de sua superclasse.

- Contudo, pode acessar os membros não `private`.

- Uma subclasse pode chamar um construtor de sua superclasse usando a palavra-chave `super` seguida por um conjunto de parênteses que contém os argumentos do construtor da superclasse.

- Isso deve aparecer como a primeira instrução no corpo do construtor da subclasse.

Relacionamento entre superclasses e subclasses

• Um método de superclasse pode ser sobrescrito em uma subclasse para declarar uma implementação apropriada para a subclasse.

• A anotação `@Override` indica que um método deve sobrescrever um método da superclasse.

• Quando o compilador encontra um método declarado com `@Override`, ele compara a assinatura do método com as assinaturas dos métodos da superclasse.

• Se não houver uma correspondência exata, o compilador emite uma mensagem de erro, como "*method does not override or implement a method from a supertype*".

Relacionamento entre superclasses e subclasses

- O método `toString` não aceita nenhum argumento e retorna uma `String`. O método `toString` da classe `Object` normalmente é sobrescrito por uma subclasse.

- Quando um objeto é enviado para a saída utilizando o especificador de formato `%s`, o método `toString` do objeto é chamado implicitamente para obter sua representação de `String`.

Construtores em subclasses

• A primeira tarefa de um construtor de subclasse é chamar o construtor de sua superclasse direta para garantir que as variáveis de instância herdadas da superclasse sejam inicializadas.

Classe `Object`

• Todas as classes em Java herdam direta ou indiretamente da classe `Object` (pacote `java.lang`), então seus 11 métodos (alguns sobrecarregados) são herdados por todas as outras classes.

Criando e utilizando uma classe CommissionEmployee

```
1  // Figura 9.4: CommissionEmployee.java
2  // A classe CommissionEmployee representa um empregado que recebeu um
3  // percentual das vendas brutas.
4  public class CommissionEmployee extends Object
5  {
6     private final String firstName;
7     private final String lastName;
8     private final String socialSecurityNumber;
9     private double grossSales; // vendas brutas semanais
10    private double commissionRate; // percentagem da comissão
11
12    // construtor de cinco argumentos
13    public CommissionEmployee(String firstName, String lastName,
14       String socialSecurityNumber, double grossSales,
15       double commissionRate)
16    {
17       // a chamada implícita para o construtor padrão de Object ocorre aqui
```

continua

continuação

```
18
19          // se grossSales é inválido, lança uma exceção
20          if (grossSales < 0.0)
21             throw new IllegalArgumentException(
22                "Gross sales must be >= 0.0");
23
24          // se commissionRate é inválido, lança uma exceção
25          if (commissionRate <= 0.0 || commissionRate >= 1.0)
26             throw new IllegalArgumentException(
27                "Commission rate must be > 0.0 and < 1.0");
28
29          this.firstName = firstName;
30          this.lastName = lastName;
31          this.socialSecurityNumber = socialSecurityNumber;
32          this.grossSales = grossSales;
33          this.commissionRate = commissionRate;
34       } // fim do construtor
35
36       // retorna o nome
37       public String getFirstName()
38       {
39          return firstName;
40       }
```

```
41
42        // retorna o sobrenome
43        public String getLastName()
44        {
45           return lastName;
46        }
47
48        // retorna o número de seguro social
49        public String getSocialSecurityNumber()
50        {
51           return socialSecurityNumber;
52        }
53
54        // configura a quantidade de vendas brutas
55        public void setGrossSales(double grossSales)
56        {
57           if (grossSales < 0.0)
58              throw new IllegalArgumentException(
59                 "Gross sales must be >= 0.0");
60
61           this.grossSales = grossSales;
62        }
63
64        // retorna a quantidade de vendas brutas
65        public double getGrossSales()
66        {
67           return grossSales;
68        }
```

```
70      // configura a taxa de comissão
71      public void setCommissionRate(double commissionRate)
72      {
73         if (commissionRate <= 0.0 || commissionRate >= 1.0)
74            throw new IllegalArgumentException(
75               "Commission rate must be > 0.0 and < 1.0");
76
77         this.commissionRate = commissionRate;
78      }
79
80      // retorna a taxa de comissão
81      public double getCommissionRate()
82      {
83         return commissionRate;
84      }
```

continua

continuação

```
85
86      // calcula os lucros
87      public double earnings()
88      {
89         return commissionRate * grossSales;
90      }
91
92      // retorna a representação String do objeto CommissionEmployee
93      @Override // indica que esse método substitui um método da superclasse
94      public String toString()
95      {
96         return String.format("%s: %s %s%n%s: %s%n%s: %.2f%n%s: %.2f",
97            "commission employee", firstName, lastName,
98            "social security number", socialSecurityNumber,
99            "gross sales", grossSales,
100           "commission rate", commissionRate);
101     }
102  } // fim da classe CommissionEmployee
```

Criando e utilizando - Classe CommissionEmployeeTest

```
1   // Figura 9.5: CommissionEmployeeTest.java
2   // Programa de teste da 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:");
15        System.out.printf("%n%s %s%n", "First name is",
16           employee.getFirstName());
17        System.out.printf("%s %s%n", "Last name is",
18           employee.getLastName());
```

continua

continuação

```
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(5000);
27          employee.setCommissionRate(.1);
28
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
```

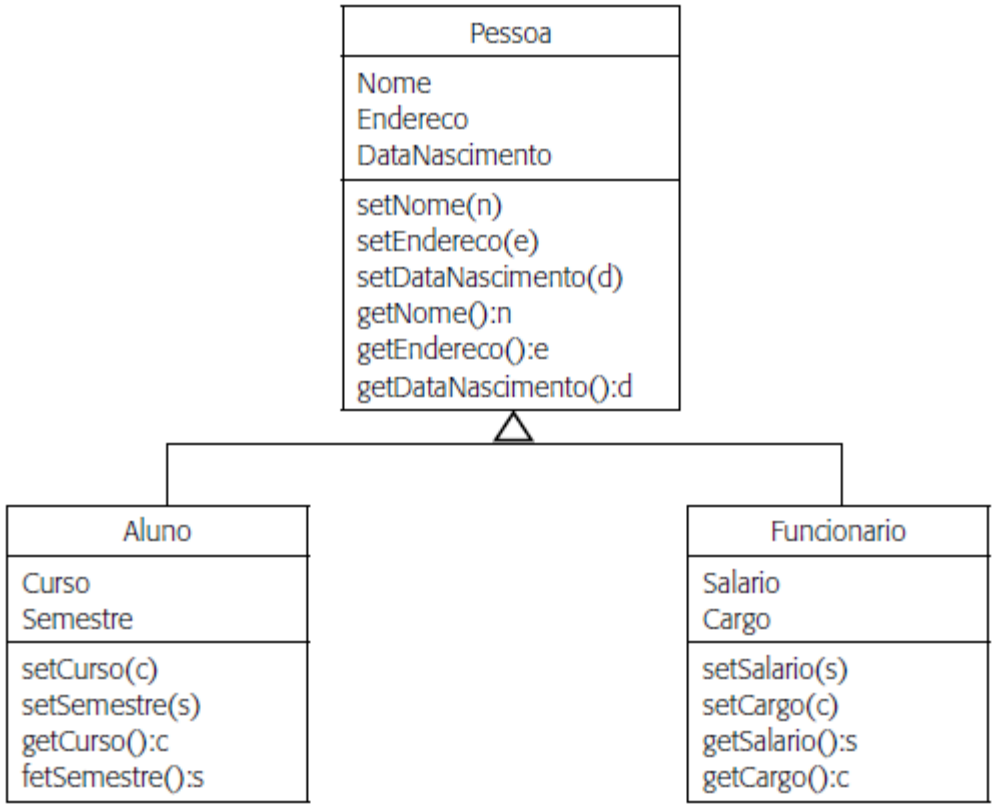
```
Employee information obtained by get methods:

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: 5000.00
commission rate: 0.10
```

Programação orientada a objetos: **Herança**



Programação orientada a objetos: Herança

- **Herança em JAVA**

- A implementação das classes descritas no diagrama em JAVA.
- Para potencializar a reutilização de código, foi criado um arquivo para cada classe, contendo a declaração dos atributos e a implementação dos métodos desejados. São eles: Pessoa.java, Aluno.java e Funcionario.java.
- java. Além disso, para demonstrar a utilização de tais classes, foi criado o arquivo Heranca.java, contendo o método main.

Referências Bibliográficas

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