Programação 2

Herança e Polimorfismo

Prof. Domingo Santos

domingos.santos@upe.br

UNIVERSIDADE DE PERNAMBUCO

Resumo

- Herança
- Polimorfismo
- Interface

Conteúdo baseado no capítulo 9 e 10 do livro de Deitel

Herança

Considerações iniciais

Herança na vida: conjunto de bens, direitos e obrigações, que uma pessoa falecida deixa aos seus sucessores.

Em programação:

- Conceito importante em OO
- Você economiza tempo durante o desenvolvimento de programas baseando novas classes existentes
- Herdar membros de uma classe existente
- **Superclasse** como a classe básica e a **subclasse** como a classe derivada

Superclasses e subclasses

- Exemplo: Sistema para manutenção de eletrônicos

Super classe

Eletronico
- marca: String - anoLancamento: int - dataEntrada: int[3] //dia mes ano
+ getMarca(): String + setMarca(String nome) + getAnoLancamento(): int + setAnoLancamento(int idade) + getDataEntrada(): int[] + setDataEntrada(int[] nome) + processNotaFiscal()

Subclasses

Computador
notebook:boolean
+ isNotebook(): boolean + setNotebook(boolean notebook)

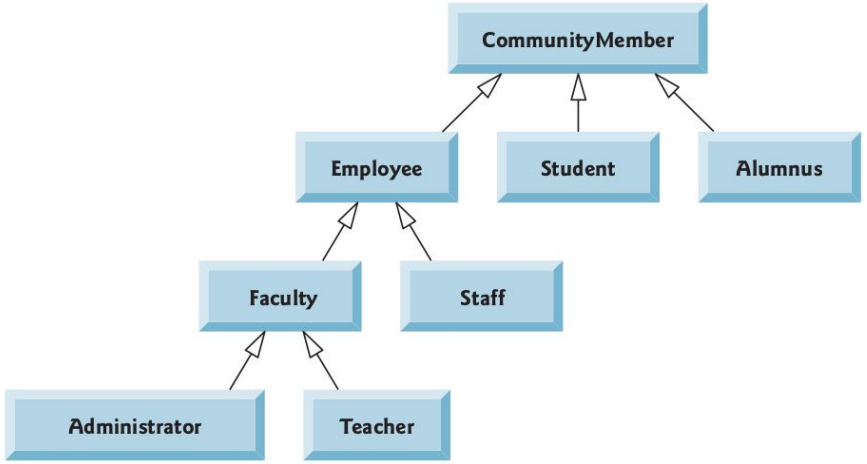
Celular

Impressora
ecoTank:boolean
+ isEcoTank(): boolean + setEcoTankboolean notebook)

Superclasses e subclasses

- Outros exemplos:

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



- O Java só suporta herança única, na qual cada classe é derivada exatamente de uma superclasse direta.
- A subclasse é um tipo específico da superclasse, mas a subclasse é diferente da superclasse
- Por exemplo, a superclasse Vehicle representa todos os veículos, incluindo carros, caminhões, barcos, bicicletas e assim por diante. Por contraste, a subclasse Car representa um subconjunto de veículos menor e mais específico.

UNIVERSIDADE DE PERNAMBUCO

Membros protected

- public:
  - são acessíveis onde quer que o programa tenha uma referência a um objeto dessa classe ou a uma de suas subclasses

- private:
  - só são acessíveis dentro da própria classe
  - permanecem ocultos de suas subclasses e só podem ser acessados por meio dos métodos public ou protected herdados da superclasse
  - caso as subclasses tivessem acesso direto os benefícios do ocultamento de informações seriam perdido

- protected:
  - nível intermediário de acesso entre public e private
  - podem ser acessados por membros dessa superclasse, de suas subclasses e de outras classes no mesmo pacote

Estudo de caso: CommissionEmployee

contém tipos de empregados no aplicativo de folha de pagamento de uma empresa

os empregados comissionados (que serão representados como objetos de uma superclasse) recebem uma porcentagem de suas vendas,

enquanto empregados comissionados com salário-base (que serão representados como objetos de uma subclasse) recebem um salário-base mais uma porcentagem de suas vendas.

Classe CommissionEmployee - part 1 (pagina 287 do livro de Deitel)

```
public class CommissionEmployee extends Object {
   // esse processo é feito internamente para toda classe que extende da classe objeto

   private final String firstName;
   private final String lastName;
   private final String socialSecurityNumber;
   private double grossSales; // vendas brutas semanais
   private double commissionRate; // percentagem da comissão
  . . .
}
```

UNIVERSIDADE DE PERNAMBUCO

Classe Object

Como vimos, toda classe estende implicitamente a classe Object. Alguns exemplos de métodos já implementados:

equals	Compara dois objetos quanto à igualdade e retorna true se eles forem iguais
toString	**A implementação padrão desse método retorna o nome do pacote e o nome da classe do objeto tipicamente seguido por uma representação hexadecimal do valor retornado pelo método hashCode do objeto**
finalize	O método protected é chamado pelo coletor de lixo para realizar limpeza de terminação sobre um objeto um pouco antes desse coletor reivindicar a memória do objeto. Deve-se evitar sobre escrever esse método
clone	Método protected, que não aceita nenhum argumento e retorna uma referência Object, faz uma cópia do objeto em que é chamado.

UNIVERSIDADE DE PERNAMBUCO

Membros protected

Classe CommissionEmployee - part 2

```
public class CommissionEmployee extends Object {
   . . .
   // construtor de cinco argumentos
   public CommissionEmployee(String firstName,
                             String lastName,
                             String socialSecurityNumber,
                             double grossSales,
                             double commissionRate) {
      // se grossSales é inválido, lança uma exceção
      if (grossSales < 0.0) {
          throw new IllegalArgumentException("Gross sales must be >= 0.0");
      }
      // se commissionRate é inválido, lança uma exceção
      if (commissionRate <= 0.0 || commissionRate >= 1.0){
          throw new IllegalArgumentException("Commission rate must be > 0.0 and < 1.0");
      }

      this.firstName = firstName;
      this.lastName = lastName;
      this.socialSecurityNumber = socialSecurityNumber;
      this.grossSales = grossSales;
      this.commissionRate = commissionRate;
   }
   . . .
}
```

UNIVERSIDADE DE PERNAMBUCO

Membros protected

Classe CommissionEmployee - part 3

```
public class CommissionEmployee extends Object {
   . . .
   // retorna o nome
   public String getFirstName()
   {
       return firstName;
   }
   // retorna o sobrenome
   public String getLastName()
   {
           return lastName;
   }
   // retorna o número de seguro social
   public String getSocialSecurityNumber()
   {
           return socialSecurityNumber;
   }
  . . .
}
```

UNIVERSIDADE DE PERNAMBUCO

Membros protected

Classe CommissionEmployee - part 4

```
public class CommissionEmployee extends Object {
   . . .
    // configura a quantidade de vendas brutas
   public void setGrossSales(double grossSales)
   {
       if (grossSales < 0.0)
           throw new IllegalArgumentException ("Gross sales must be >= 0.0" );
       this.grossSales = grossSales;
   }
   // retorna a quantidade de vendas brutas
   public double getGrossSales()
   {
       return grossSales;
   }
   // configura a taxa de comissão
   public void setCommissionRate(double commissionRate)
   {
       if (commissionRate <= 0.0 || commissionRate >= 1.0)
                   throw new IllegalArgumentException ( "Commission rate must be > 0.0 and < 1.0" );
       this.commissionRate = commissionRate;
   }
   // retorna a taxa de comissão
   public double getCommissionRate()
   {
       return commissionRate;
   }
  . . .
}
```

UNIVERSIDADE DE PERNAMBUCO

Membros protected

Classe CommissionEmployee - part 5

```
public class CommissionEmployee extends Object {
   . . .
   // calcula os lucros
   public double earnings()
   {
       return commissionRate * grossSales;
   }
   // retorna a representação String do objeto CommissionEmployee
   @Override // indica que esse método substitui um método da superclasse
   public String toString()
   {
       return String.format("%s: %s %s%n%s: %s%n%s: %.2f%n%s: %.2f",
                   "commission employee", firstName, lastName,
                   "social security number", socialSecurityNumber,
                   "gross sales", grossSales,
                   "commission rate", commissionRate);
   }

}
```

UNIVERSIDADE DE PERNAMBUCO

Membros protected

Classe BasePlusCommissionEmployee - parte 1

```
public class BasePlusCommissionEmployee extends CommissionEmployee {
   private double baseSalary; // salário-base por semana

   public BasePlusCommissionEmployee(String firstName, String lastName,
          String socialSecurityNumber, double grossSales,
          double commissionRate, double baseSalary){
      // chamada explícita para o construtor CommissionEmployee da superclasse
      super(firstName, lastName, socialSecurityNumber,
      grossSales, commissionRate);

      // se baseSalary é inválido, lança uma exceção
      if (baseSalary < 0.0){
          throw new IllegalArgumentException( "Base salary must be >= 0.0");
      }

      this.baseSalary = baseSalary;
   }
  . . .
}
```

UNIVERSIDADE DE PERNAMBUCO

Membros protected

Classe BasePlusCommissionEmployee - parte 2

```
public class BasePlusCommissionEmployee extends CommissionEmployee {
   . . .
  // configura o salário-base
   public void setBaseSalary(double baseSalary)
   {
       if (baseSalary < 0.0){
           throw new IllegalArgumentException("Base salary must be >= 0.0");
       }
       this.baseSalary = baseSalary;
   }
   // retorna o salário-base
   public double getBaseSalary()
   {
       return baseSalary;
   }

  . . .
}
```

Membros protected

Classe BasePlusCommissionEmployee - parte 3

```
public class BasePlusCommissionEmployee extends CommissionEmployee {
   . . .
// calcula os lucros
   @Override
   public double earnings()
   {
       return getBaseSalary() + super.earnings();
   }


   // retorna a representação de String de BasePlusCommissionEmployee
   @Override
   public String toString()
   {
       // não permitido: tenta acessar membros private da superclasse
       return String.format(
       "%s: %s %s%n%s: %s%n%s: %.2f%n%s: %.2f%n%s: %.2f",
       "base-salaried commission employee", firstName, lastName,
       "social security number", socialSecurityNumber,
       "gross sales", grossSales, "commission rate", commissionRate,
       "base salary", baseSalary);
   }
   . . .
}
```

Sobrescrevendo métodos da super class

Qual é o problema no método toString ?

tenta acessar membros private da superclasse

Membros protected

Classe BasePlusCommissionEmployee - opções de contorno para o problema do slide anterior

Por para protected na super classe

```
public class CommissionEmployee extends Object
{
   protected final String firstName;
   protected final String lastName;
   protected final String socialSecurityNumber;
   protected double grossSales;
   protected double commissionRate;
  . . .
}
```

chamar métodos get da super classe

```
public class BasePlusCommissionEmployee extends CommissionEmployee {
  . . .
   @Override
   public String toString()
   {
       return String.format(
       "%s: %s %s%n%s: %s%n%s: %.2f%n%s: %.2f%n%s: %.2f",
       "base-salaried commission employee", getFirstName(), getLastName(),
       "social security number", getSocialSecurityNumber(),
       "gross sales", getBaseSalary(),
       "commission rate", getCommissionRate(),
       "base salary", getBaseSalary());
   }
  . . .
}
```

Membros protected

Chamando subclasse no .java main

```
public class BasePlusCommissionEmployeeTest{
   public static void main(String[] args)
   {
      // instancia o objeto BasePlusCommissionEmployee
      BasePlusCommissionEmployee employee =
      new BasePlusCommissionEmployee("Bob", "Lewis", "333-33-3333", 5000, .04, 300);

      // obtém os dados do empregado comissionado com salário-base
      System.out.println("Employee information obtained by get methods:%n");
      System.out.printf("%s %s%n", "First name is",
                        employee.getFirstName());
      System.out.printf("%s %s%n", "Last name is",
                        employee.getLastName());
      System.out.printf("%s %s%n", "Social security number is",
                        employee.getSocialSecurityNumber());
      System.out.printf("%s %.2f%n", "Gross sales is",
                        employee.getGrossSales());
      System.out.printf("%s %.2f%n", "Commission rate is",
                        employee.getCommissionRate());
      System.out.printf("%s %.2f%n", "Base salary is",
                        employee.getBaseSalary());

      employee.setBaseSalary(1000);

      System.out.printf("%n%s:%n%n%s%n", "Updated information",
                        employee.toString());
   } // fim de main
}
```

```
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 toStr

base-salaried commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: 1000,00
commission rate: 0,04
base salary: 1000,00
BUILD SUCCESSFUL (total time: 0 seconds)
```

Prática !

Crie a super classe Veiculo, e deve ter as seguintes subclasses: Carro, Moto e Caminhao. Deve existir os métodos get e set.

- Veiculo:
  - atributos:
    - String marca;
    - int ano;
  - método:
    - acelerar, sem retorno, exibir na tela “Veículo acelerando...”
- Carro:
  - atributo:
    - int numeroPortas;
  - método:
    - ligarRadio, sem retorno exibir na tela “Rádio ligado no carro.”
- Caminhão:
  - atributo:
    - float carga máxima;
  - método:
    - carregarCarga

- Crie o .java final e chame todas as classes, ao lado temos um exemplo

```
public class Main {

    public static void main(String[] args) {
        Carro carro = new Carro("Ford", 2022, 4);
        Caminhao caminhao = new Caminhao("Volvo", 2020,
15000);

        carro.acelerar();
        carro.ligarRadio();

        caminhao.acelerar();
        caminhao.carregarCarga();
    }
}
```

UNIVERSIDADE DE PERNAMBUCO

Polimorfismo

Considerações Iniciais

Muitas formas

Permite que objetos de diferentes classes sejam tratados de forma uniforme

Flexibilidade e reutilização de código em Java

Sobrecarga: permite criar métodos com funcionalidades similares, mas que aceitam diferentes tipos de entrada, implementação na mesma classe

Pode ser utilizado para modificar o comportamento de métodos em classes filhas, mantendo a mesma assinatura

Permite escrever programas que processam objetos que compartilham a mesma superclasse, direta ou indiretamente, **como se todos fossem objetos da superclasse**

UNIVERSIDADE DE PERNAMBUCO

Exemplo

```java
public class Animal {
    public void fazerSom() {
        System.out.println("Som genérico de animal");
    }
}

public class Cachorro extends Animal {
    @Override
    public void fazerSom() {
        System.out.println("latir!");
    }
}


public class Gato extends Animal {
    @Override
    public void fazerSom() {
        System.out.println("miar!");
    }
}
```

Classe animal e subclasses

Não seria necessário conhecimento específico para utilizar subclasses de Animal

UNIVERSIDADE DE PERNAMBUCO

Exemplos

Diferentes classes tratadas de forma uniforme

```
public class PolymorphismTest {

   public static void main(String[] args) {
       CommissionEmployee commissionEmployee = new CommissionEmployee(
               "Sue", "Jones", "222-22-2222", 10000, 0.06);

       CommissionEmployee basePlusCommissionEmployee
               = new BasePlusCommissionEmployee(
                       "Bob", "Lewis", "333-33-3333", 5000, 0.04, 300);

       System.out.printf("%s\n\n\n", commissionEmployee.toString());
       System.out.printf("%s\n", basePlusCommissionEmployee.toString());

   } // fim de main
} // fim da classe PolymorphismTest
```

```
commission employee: Sue Jones
social security number: 222-22-2222
gross sales: 10000,00
commission rate: 0,06


base-salaried commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: 300,00
commission rate: 0,04
base salary: 300,00
```

Exemplos

Diferentes classes tratadas de forma uniforme

```
public class PolymorphismTest {

   public static void main(String[] args) {
       CommissionEmployee commissionEmployee = new CommissionEmployee( "Sue", "Jones",
                                                                       "222-22-2222", 10000, 0.06);

       CommissionEmployee basePlusCommissionEmployee = new BasePlusCommissionEmployee(
                       "Bob", "Lewis", "333-33-3333", 5000, 0.04, 300);

       System.out.printf("%s\n\n\n", commissionEmployee.toString());
       System.out.printf("%s\n", basePlusCommissionEmployee.toString());
       System.out.printf("%s\n", basePlusCommissionEmployeetgetBaseSalary());
   } // fim de main
} // fim da classe PolymorphismTest
```

```
Exception in thread "main" java.lang.RuntimeException: Uncompilable code - cannot find symbol
  symbol:   method getBaseSalary()
  location: variable basePlusCommissionEmployee of type methods.CommissionEmployee
        at methods.PolymorphismTest.main(PolymorphismTest.java:1)
/home/domingossj/.cache/netbeans/21/executor-snippets/run.xml:111: The following error occurred while executing this line:
/home/domingossj/.cache/netbeans/21/executor-snippets/run.xml:68: Java returned: 1
BUILD FAILED (total time: 2 seconds)
```

UNIVERSIDADE DE PERNAMBUCO

Classes e métodos abstratos

Classe abstrata:

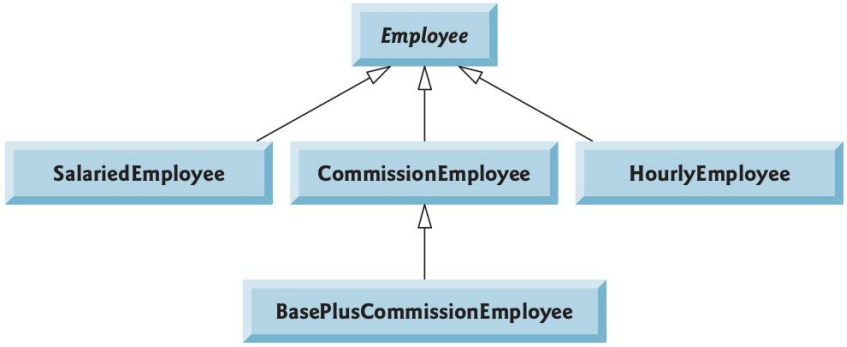
- Superclasses em hierarquias de herança, são chamadas superclasses abstratas
- Não podem ser chamadas de forma direta
- O propósito de uma classe abstrata é fornecer uma superclasse apropriada a partir da qual outras classes podem herdar e assim compartilhar um design comum
- Superclasses abstratas são excessivamente gerais para criar objetos reais
- Só especificam o que é comum entre subclasses
- Tentar instanciar um objeto de uma classe abstrata é um erro de compilação.

Método abstrato:

- Exemplo: public abstract void draw()
- Podem aparecer dentro de classes abstratas ou interfaces
- Não fornece implementações. Fica obrigatório a implementação pela subclasse
- Não pode ser static

Sistema de folha de pagamento

Arquitetura do sistema



	earnings	toString
Employee	abstract	*firstName lastName* social security number: *SSN*
Salaried-Employee	weeklySalary	salaried employee: *firstName lastName* social security number: *SSN* weekly salary: *weeklySalary*

Hourly-Employee	if (hours <= 40)
	  wage * hours
	else if (hours > 40)
	{
	  40 * wage +
	  (hours - 40) *
	  wage * 1.5
	}
		hourly employee: *firstName lastName*
		social security number: *SSN*
		hourly wage: *wage*; hours worked: *hours*

Commission-Employee	commissionRate * grossSales	commission employee: *firstName lastName* social security number: *SSN* gross sales: *grossSales*; commission rate: *commissionRate*
BasePlus-Commission-Employee	(commissionRate * grossSales) + baseSalary	base salaried commission employee: *firstName lastName* social security number: *SSN* gross sales: *grossSales*; commission rate: *commissionRate*; base salary: *baseSalary*

UNIVERSIDADE DE PERNAMBUCO

Sistema de folha de pagamento

Classe abstrata Employee pt 1 (Página 320)

```
public abstract class Employee {

   private final String firstName;
   private final String lastName;
   private final String socialSecurityNumber;

   public Employee(String firstName, String lastName, String socialSecurityNumber) {
       this.firstName = firstName;
       this.lastName = lastName;
       this.socialSecurityNumber = socialSecurityNumber;
   }
   public String getFirstName() {
       return firstName;
   }
   public String getLastName() {
       return lastName;
   }
   public String getSocialSecurityNumber() {
       return socialSecurityNumber;
   }
  .  .  .
}
```

Sistema de folha de pagamento

Classe abstrata Employee pt 2 (Página 320)

```
public abstract class Employee {
   . . .
   @Override
   public String toString() {
       return String.format("%s %s%nsocial security number: %s", getFirstName(),
getLastName(), getSocialSecurityNumber());
   }

   public abstract double earnings(); // nenhuma implementação aqui
}
```

Sistema de folha de pagamento

Classe SalariedEmployee pt 1 (Página 321)

```
public class SalariedEmployee extends Employee {

   private double weeklySalary;

   // construtor
   public SalariedEmployee(String firstName, String lastName,
                           String socialSecurityNumber, double weeklySalary) {
       super(firstName, lastName, socialSecurityNumber);
       if (weeklySalary < 0.0) {
           throw new IllegalArgumentException("Weekly salary must be >= 0.0");
       }
       this.weeklySalary = weeklySalary;
   }
  . . .
}
```

Sistema de folha de pagamento

Classe SalariedEmployee pt 2 (Página 321)

```
public class SalariedEmployee extends Employee {
    . . .
    public void setWeeklySalary(double weeklySalary) {
        if (weeklySalary < 0.0) {
            throw new IllegalArgumentException("Weekly salary must be >= 0.0");
        }
        this.weeklySalary = weeklySalary;
    }
    public double getWeeklySalary() {
        return weeklySalary;
    }
    @Override
    public double earnings() {
        return getWeeklySalary();
    }
    @Override
    public String toString() {
        return String.format("salaried employee: %s%n%s: $%,.2f", super.toString(),
                             "weekly salary", getWeeklySalary());
    }
}
```

Sistema de folha de pagamento

Classe HourlyEmployee pt 1 (Página 322)

```
public class HourlyEmployee extends Employee {

   private double wage; // salário por hora
   private double hours; // horas trabalhadas durante a semana

   public HourlyEmployee(String firstName, String lastName, String socialSecurityNumber, double
wage, double hours) {
       super(firstName, lastName, socialSecurityNumber);
       if (wage < 0.0) {
           throw new IllegalArgumentException("Hourly wage must be >= 0.0");
       }
       if ((hours < 0.0) || (hours > 168.0)) {
           throw new IllegalArgumentException(
                   "Hours worked must be >= 0.0 and <= 168.0");
       }
       this.wage = wage;
       this.hours = hours;
   }
  . . .
}
```

Sistema de folha de pagamento

Classe HourlyEmployee pt 2 (Página 322)

```
public class HourlyEmployee extends Employee {
    . . .
    public void setWage(double wage) {
        if (wage < 0.0) {
            throw new IllegalArgumentException("Hourly wage must be >= 0.0");
        }
        this.wage = wage;
    }
    public double getWage() {
        return wage;
    }
    public void setHours(double hours) {
        if ((hours < 0.0) || (hours > 168.0)) {
            throw new IllegalArgumentException("Hours worked must be >= 0.0 and <= 168.0");
        }
        this.hours = hours;
    }
    public double getHours() {
        return hours;
    }
    . . .
}
```

Sistema de folha de pagamento

Classe HourlyEmployee pt 3 (Página 322)

```
public class HourlyEmployee extends Employee {
    . . .
    @Override
    public double earnings() {
        if (getHours() <= 40) // nenhuma hora extra
        {
            return getWage() * getHours();
        } else {
            return 40 * getWage() + (getHours() - 40) * getWage() * 1.5;
        }
    }

    @Override
    public String toString() {
        return String.format("hourly employee: %s%n%s: $%,.2f; %s: %,.2f", super.toString(), "hourly
wage", getWage(), "hours worked", getHours());
    }
}
```

Sistema de folha de pagamento

Classe CommissionEmployee pt 1 (Página 324)

```
public class CommissionEmployee extends Employee {

   private double grossSales; // vendas brutas semanais
   private double commissionRate; // porcentagem da comissão

   public CommissionEmployee(String firstName,
                  String lastName,
                  String socialSecurityNumber,
                  double grossSales,
                  double commissionRate) {

      super(firstName, lastName, socialSecurityNumber);

      if (commissionRate <= 0.0 || commissionRate >= 1.0) {
          throw new IllegalArgumentException("Commission rate must be > 0.0 and < 1.0");
      }
      if (grossSales < 0.0) {
          throw new IllegalArgumentException("Gross sales must be >= 0.0");
      }
      this.grossSales = grossSales;
      this.commissionRate = commissionRate;
   }
  . . .
}
```

Sistema de folha de pagamento

Classe CommissionEmployee pt 2 (Página 324)

```
public class CommissionEmployee extends Employee {
    . . .
    public void setGrossSales(double grossSales) {
        if (grossSales < 0.0) {
            throw new IllegalArgumentException("Gross sales must be >= 0.0");
        }
        this.grossSales = grossSales;
    }

    public double getGrossSales() {
        return grossSales;
    }

    public void setCommissionRate(double commissionRate) {
        if (commissionRate <= 0.0 || commissionRate >= 1.0) {
            throw new IllegalArgumentException("Commission rate must be > 0.0 and < 1.0");
        }
        this.commissionRate = commissionRate;
    }
    . . .
}
```

Sistema de folha de pagamento

Classe CommissionEmployee pt 3 (Página 324)

```
public class CommissionEmployee extends Employee {
    . . .
    public double getCommissionRate() {
        return commissionRate;
    }

    @Override
    public double earnings() {
        return getCommissionRate() * getGrossSales();
    }

    @Override
    public String toString() {
        return String.format("%s: %s%n%s: $%,.2f; %s: %.2f",
                "commission employee", super.toString(),
                "gross sales", getGrossSales(),
                "commission rate", getCommissionRate());
    }
}
```

Sistema de folha de pagamento

Classe BasePlusCommissionEmployee pt 1 (Página 325)

```
public class BasePlusCommissionEmployee extends CommissionEmployee {

   private double baseSalary; // salário de base por semana

   public BasePlusCommissionEmployee(String firstName, String lastName,
           String socialSecurityNumber, double grossSales,
           double commissionRate, double baseSalary) {

       super(firstName, lastName, socialSecurityNumber, grossSales, commissionRate);
       if (baseSalary < 0.0) {
           throw new IllegalArgumentException("Base salary must be >= 0.0");
       }
       this.baseSalary = baseSalary;
   }
   . . .
}
```

Sistema de folha de pagamento

Classe BasePlusCommissionEmployee pt 2 (Página 325)

```
public class BasePlusCommissionEmployee extends CommissionEmployee {
    . . .
    public void setBaseSalary(double baseSalary) {
        if (baseSalary < 0.0){
            throw new IllegalArgumentException("Base salary must be >= 0.0");
        }
        this.baseSalary = baseSalary;
    }

    public double getBaseSalary(){
        return baseSalary;
    }
    @Override
    public double earnings(){
        return getBaseSalary() + super.earnings();
    }
    @Override
    public String toString(){
        return String.format("%s %s; %s: $%,.2f", "base-salaried", super.toString(),
                             "base salary", getBaseSalary());
    }
}
```

Sistema de folha de pagamento

Classe main pt 1 (Página 326)

```
public class PayrollSystemTest {
   public static void main(String[] args) {
      SalariedEmployee salariedEmployee = new SalariedEmployee("John", "Smith", "111-11-1111", 800.00);
      HourlyEmployee hourlyEmployee = new HourlyEmployee("Karen", "Price", "222-22-2222", 16.75, 40);
      CommissionEmployee commissionEmployee = new CommissionEmployee("Sue", "Jones", "333-33-3333", 10000, .06);
      BasePlusCommissionEmployee basePlusCommissionEmployee = new BasePlusCommissionEmployee("Bob", "Lewis",
"444-44-4444", 5000, .04, 300);

      System.out.println("Employees processed individually:");
      System.out.printf("%n%s%n%s: $%,.2f%n%n", salariedEmployee, "earned", salariedEmployee.earnings());
      System.out.printf("%s%n%s: $%,.2f%n%n", hourlyEmployee, "earned", hourlyEmployee.earnings());
      System.out.printf("%s%n%s: $%,.2f%n%n", commissionEmployee, "earned", commissionEmployee.earnings());
      System.out.printf("%s%n%s: $%,.2f%n%n", basePlusCommissionEmployee, "earned",
basePlusCommissionEmployee.earnings());
   }
  . . .
}
```

```
Employees processed individually:

salaried employee: John Smith
social security number: 111-11-1111
weekly salary: $800,00
earned: $800,00

hourly employee: Karen Price
social security number: 222-22-2222
hourly wage: $16,75; hours worked: 40,00
earned: $670,00

commission employee: Sue Jones
social security number: 333-33-3333
gross sales: $10.000,00; commission rate: 0,06
earned: $600,00

base-salaried commission employee: Bob Lewis
social security number: 444-44-4444
gross sales: $5.000,00; commission rate: 0,04; base salary: $300,00
earned: $500,00
```

Sistema de folha de pagamento

Classe main pt 2 (Página 326)

```
public class PayrollSystemTest {
   public static void main(String[] args) {
      Employee[] employees = new Employee[4];
      employees[0] = salariedEmployee;
      employees[1] = hourlyEmployee;
      employees[2] = commissionEmployee;
      employees[3] = basePlusCommissionEmployee;
      // processa genericamente
      for (Employee currentEmployee : employees)
         System.out.println(currentEmployee);

         if (currentEmployee instanceof BasePlusCommissionEmployee) {
            // downcast da referência de Employee para referência BasePlusCommissionEmployee
            BasePlusCommissionEmployee employee = (BasePlusCommissionEmployee) currentEmployee;
            employee.setBaseSalary(1.10 * employee.getBaseSalary());
            System.out.printf("new base salary with 10%% increase is: $%,.2f%n" , employee.getBaseSalary());
         }
         System.out.printf("earned $%,.2f%n%n", currentEmployee.earnings());
      }
   . . .
   }
}
```

```
salaried employee: John Smith
social security number: 111-11-1111
weekly salary: $800,00
earned $800,00

hourly employee: Karen Price
social security number: 222-22-2222
hourly wage: $16,75; hours worked: 40,00
earned $670,00

commission employee: Sue Jones
social security number: 333-33-3333
gross sales: $10.000,00; commission rate: 0,06
earned $600,00

base-salaried commission employee: Bob Lewis
social security number: 444-44-4444
gross sales: $5.000,00; commission rate: 0,04; base salary: $300,00
new base salary with 10% increase is: $330,00
earned $530,00
```

UNIVERSIDADE DE PERNAMBUCO

Sistema de folha de pagamento

Classe main pt 2 (Página 326)

```
public class PayrollSystemTest {
   public static void main(String[] args) {
       . . .
       for (int j = 0; j < employees.length; j++) {
           System.out.printf("Employee %d is a %s%n", j, employees[j].getClass().getName());
       }

   }
}
```

```
Employee 0 is a methods.SalariedEmployee
Employee 1 is a methods.HourlyEmployee
Employee 2 is a methods.CommissionEmployee
Employee 3 is a methods.BasePlusCommissionEmployee
BUILD SUCCESSFUL (total time: 0 seconds)
```

UNIVERSIDADE DE PERNAMBUCO

Métodos e classes final

Método final:

- Não podem ser sobrescritos
- Um método final em uma superclasse não pode ser sobrescrito como uma subclasse
- Métodos que são declarados private são implicitamente final, porque não é possível sobrescrevê-los como uma subclasse.
- Métodos que são declarados static também são implicitamente final
- Uma declaração do método final nunca pode mudar, assim todas as subclasses utilizam a mesma implementação do método

UNIVERSIDADE DE PERNAMBUCO

Métodos e classes final

Classe final

- Não pode ser estendida para criar uma subclasse
- Todos os métodos em uma classe final são implicitamente final
- String é um exemplo de uma classe final
- Tornar a classe final também
- Impede que programadores criem subclasses que poderiam driblar as restrições de segurança
- Compiladores podem executar várias otimizações quando sabem que algo é final
- Tentar declarar uma subclasse de uma classe final é um erro de compilação.

Interface

UNIVERSIDADE DE PERNAMBUCO

Considerações iniciais

Uma interface em Java é uma coleção de métodos abstratos e constantes públicas

Define um contrato que as classes que a implementam devem seguir, especificando quais métodos devem ser implementados, mas não fornecendo a implementação desses métodos

Uma declaração de interface inicia com a palavra-chave interface e contém somente constantes e métodos abstract

devem ser public e as interfaces não podem especificar nenhum detalhe de implementação, como declarações de método concretas e variáveis de instância

Para especificar que uma classe implementa uma interface, adicione a palavra-chave implements e o nome da interface

Funciona como assinar um contrato com o compilador que afirma, “Irei declarar todos os métodos especificados pela interface ou irei declarar minha classe abstract”

Falhar em implementar qualquer método de uma interface em uma classe concreta que implementa a interface resulta em um erro de compilação indicando que a classe deve ser declarada abstract

Considerações iniciais

- criação: public interface NomeInterface
- Permite que objetos de classes não relacionadas sejam processados polimorficamente
- objetos de classes que implementam a mesma interface podem responder às mesmas chamadas de método
- Diferenças:

Interface	**Classe abstrata**
Não possuem atributos	Possuem atributos
Apenas métodos abstratos	Pode possuir métodos abstratos e concretos
Não pode declarar corpo nos métodos	Pode declarar corpo nos métodos
Não pode ser instanciada	Não pode ser instanciada

Considerações iniciais

Exemplo: interface pagamento

```java
// Definição da interface Pagamento
public interface Pagamento {
    void processarPagamento(double valor);
}
// Implementação do pagamento com cartão de crédito
public class CartaoCredito implements Pagamento {
    @Override
    public void processarPagamento(double valor) {
        System.out.println("Processando pagamento com cartão de crédito no valor de R$" + valor);
        // Lógica específica para processamento de pagamento com cartão de crédito
    }
}
// Implementação do pagamento via PayPal
public class PayPal implements Pagamento {
    @Override
    public void processarPagamento(double valor) {
        System.out.println("Processando pagamento via PayPal no valor de R$" + valor);
        // Lógica específica para processamento de pagamento via PayPal
    }
}
```