



UNIVERSIDADE
FEDERAL DO CEARÁ

TÓPICO
10



PROGRAMAÇÃO ORIENTADA A OBJETOS

Prof. Marcos Vinicius de Andrade Lima
E-mail: marcos.vinicius@ufc.br



Olá!

Sou Marcos Vinicius

No tópico passado nós aplicamos o mecanismo de herança com Java...

Neste tópico aprenderemos como **redefinir comportamentos!**

“

**Não reze por uma vida fácil, reze por
forças para suportar uma difícil**
(Bruce Lee)



Introdução

VOCÊ SACA DOS PARANAUÊ?

```
public class Robot{
    ...
    public static int totalRobots;
    public Robot(){ totalRobots++; }
    ...}

```

```
public class RobotWeb extends Robot{
    ...
    public RobotWeb() { totalRobots++; }
    ...}

```

```
public class TesteRobots{
    public static void main(String[] args){
        Robot r2d2 = new Robot();
        RobotWeb spider = new RobotWeb();
        Sys...println(Robot.totalRobots + " " +
                        RobotWeb.totalRobots);    } }

```

Qual valor
será exibido
na tela?



Prof. Marcos Vinicius – UFC/Russas - POO

5/37

FILHO DE PEIXE, PEIXINHO É?

Luke, eu sou
seu pai!

Sai fora!



Prof. Marcos Vinicius – UFC/Russas - POO

6/37



Redefinição de Métodos

INTRODUÇÃO

- Algumas vezes a subclasse precisa alterar o comportamento que foi herdado. Por exemplo, imagine uma classe **ContaCorrente**:

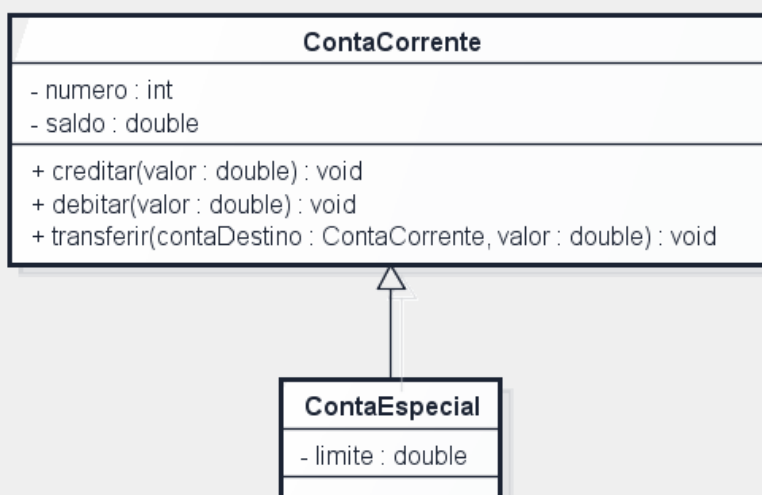
```
class ContaCorrente {  
    int numero;  
    double saldo;  
    public void creditar(double valor){  
        ...  
    }  
    public void debitar(double valor){  
        if ( valor <= saldo ) {  
            saldo -= valor; }  
    }  
}
```

Sr. e Sra. Silva, sua **Conta Especial** conjunta foi aprovada! Com ela terão um **limite maior** de crédito, além de diversas **outras facilidades!**

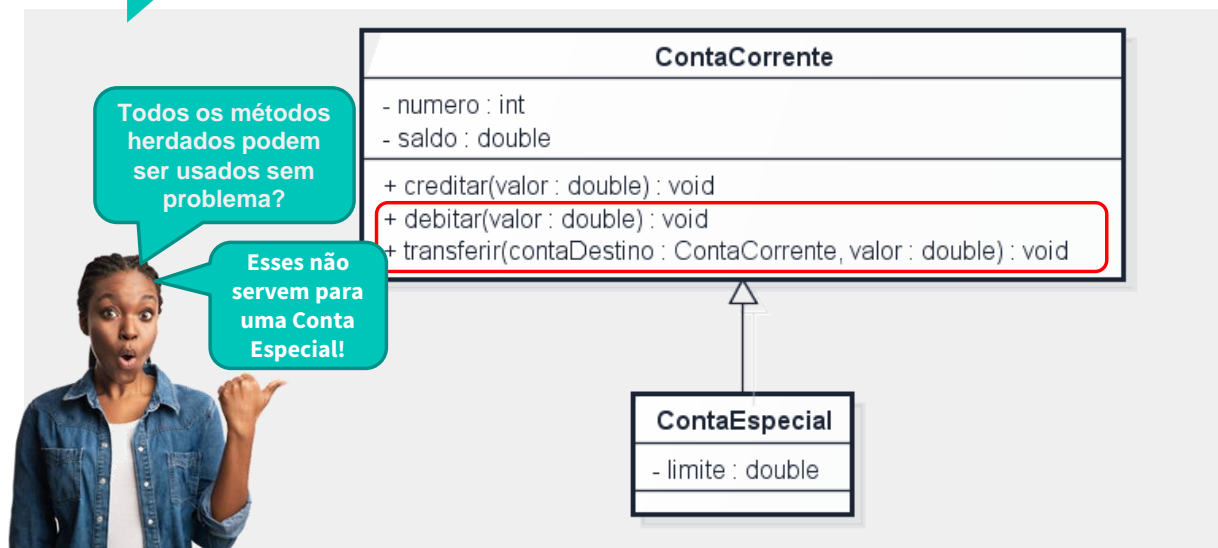


VEJA BEM...

Herança está ficando bem interessante!



VEJA BEM...



Prof. Marcos Vinicius – UFC/Russas - POO

11/37

REDEFINIÇÃO DE MÉTODOS

- O método **debitar()** de **ContaCorrente** não serve para **ContaEspecial**, embora seja herdado!
- Para **especializar o comportamento** de uma classe, muitas vezes se faz necessário **redefinir métodos em subclasses**. Uma forma de fazer isso é através da **redefinição ou sobrescrita (*overriding*)** de métodos.

Prof. Marcos Vinicius – UFC/Russas - POO

12/37



CUIDADO!!!

A **redefinição de métodos** ocorre quando uma classe define um método usando o **mesmo nome, tipo de retorno e argumentos** de um **método da superclasse**.

REDEFINIÇÃO DE MÉTODOS

- Um método sobrescrito é um método na subclasse com a **mesma assinatura**:
 - ✓ mesmo nome de método;
 - ✓ mesmos parâmetros (mesmos nomes, mesmos tipos, mesma quantidade e mesma ordem);
 - ✓ mesmo tipo de retorno de um método na superclasse.
- O que muda é somente a implementação que será diferente na subclasse.

REDEFINIÇÃO DE MÉTODOS

- A **semântica e a visibilidade** dos métodos redefinidos **deve ser preservada** (a **visibilidade**, entretanto, **pode ser aumentada**).

private ➡ package ➡ protected ➡ public



Prof. Marcos Vinicius – UFC/Russas - POO

15/37



A caixa de **Pandora** foi aberta...

APLICANDO A REDEFINIÇÃO DE MÉTODO

```
class ContaCorrente {
    int numero;
    double saldo;
    public void creditar( double valor ) { ... }
    public void debitar( double valor ) {
        if ( valor <= saldo ) saldo -= valor;
    }
}

class ContaEspecial extends ContaCorrente {
    private double limite;
    public void debitar( double valor ) {
        if ( valor <= saldo + limite ) {
            saldo = (valor > saldo) ? 0 : saldo-valor;
            limite -= (valor - saldo);
        }
    }
    public void transferir(ContaCorrente conta, double valor){
        ...
    }
}
```

Prof. Marcos Vinicius – UFC/Russas - POO

17/37

OUTRO EXEMPLO RÁPIDO PARA FIXAR

O método **f()** da classe **A** abaixo está redefinido (ou sobrescrito) na subclasse **B**:

```
class A {
    void f( ) {
        System.out.println ("Classe A");
    }
}

class B extends A{
    void f( ) {
        System.out.println ("Classe B");
    }
}
```

Prof. Marcos Vinicius – UFC/Russas - POO

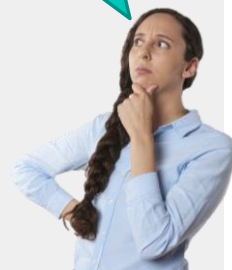
18/37

RESPONDA AGORA...

Com base nas classes anteriores, o que será mostrado na tela se a classe abaixo for executada?

```
public class TesteSobrescrita1 {
    public static void main ( String args [ ] ) {
        B b = new B ();
        b.f();
        A a = new A();
        a.f();
        a = b;
        a.f();
    }
}
```

Calma aí...
Pensando!



A classe do objeto (e não da variável de referência) determina o método a ser executado.

Prof. Marcos Vinicius – UFC/Russas - POO

19/37

VAMOS PENSAR UM POUCO MAIS...

- Imagine que existe uma regra que diz que um estudante tutor **não** pode ter nenhuma nota abaixo de **7.0**. Caso ele tenha, uma mensagem deve ser exibida informando que ele perdeu a bolsa.
- **Então, o que deve ser feito?**
 - ✓ Deve-se empregar a redefinição do método **atribuirNota()** na classe **EstudanteTutor()**.
 - ✓ O método **atribuirNota()** tem que ser modificado para incluir o teste da nota, teste este que não era necessário para os estudantes comuns. Então, estamos, assim, especializando o método **atribuirNota()**.

Prof. Marcos Vinicius – UFC/Russas - POO

20/37

SÓ SEI QUE FOI ASSIM...

```
class Estudante {  
    ...  
    public void atribuirNota( int numProva, double nota) {  
        notas[numProva-1] = nota; }  
    }  
  
class EstudanteTutor extends Estudante {  
    ...  
    public void atribuirNota( int numProva, double nota) {  
        if (nota >= 7.0)  
            notas[numProva-1] = nota;  
        else  
            System.out.println("Tutor desligado!"); }  
    }
```

mesmo tipo de
retorno e
assinatura

Prof. Marcos Vinicius – UFC/Russas - POO

21/37



A Referência super

A REFERÊNCIA SUPER

- A palavra-chave **super** pode ser usada no corpo de um método de instância em uma subclasse para acessar variáveis e invocar métodos herdados da superclasse.
- A palavra-chave **super** provê uma referência ao objeto corrente como uma instância de sua superclasse.
- É geralmente usada para invocar métodos sobrescritos (**overridden methods**) ou acessar variáveis que foram ocultadas pela subclasse (*shadowed variables*).
- **Só é possível acessar a definição dos métodos da superclasse imediata.**

Chocada!



Prof. Marcos Vinicius – UFC/Russas - POO

23/37

UM EXEMPLO PARA ANIMAR...

```
class A {
    void f( ) {
        System.out.println( "Classe A" );
    }
}

class B extends A {
    void f( ) {
        super.f();
        System.out.println( "Classe B" );
    }
}
```

Prof. Marcos Vinicius – UFC/Russas - POO

24/37

VAMOS BRINCAR NOVAMENTE?

```
public class TesteSobrescrita {
    public static void main (String args [ ])
    {
        B b = new B ();
        b.f();          // Imprime "Classe A" e "Classe B"
        A a = new A();
        a.f();          // Imprime "Classe A"
        a = b;
        a.f();          // Imprime "Classe A" e "Classe B"
    }
}
```

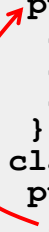
Prof. Marcos Vinicius – UFC/Russas - POO

25/37

PARA FIXAR A REFERÊNCIA SUPER

- No exemplo do estudante, o método `exibir` poderia ser redefinido em **EstudanteTutor** para imprimir as novas propriedades do tutor.

```
class Estudante {
    public void exibir() {
        System.out.println("matricula="+matricula);
        System.out.println("nome="+nome);
        System.out.println("sexo="+sexo);
    }
}
class EstudanteTutor extends Estudante {
    public void exibir() {
        super.exibir();
        System.out.println("bolsa="+bolsa);
        System.out.println("disc.="+ disciplina);
    }
}
```



Prof. Marcos Vinicius – UFC/Russas - POO

26/37



O Construtor `super()`

MECANISMO DA HERANÇA EM AÇÃO

- Construtores **não são herdados**, mas podem ser acessados.
- A chamada **`super()`** invoca um construtor na **superclasse imediata**, com os mesmos parâmetros da chamada.
- Assim como a chamada **`this()`**, a chamada **`super()`** só pode ser usada em **definições de construtores**.
- A chamada **`super()`**, quando usada, deve sempre ser a **primeira** sentença no código do construtor.

OLHA SÓ AS IDEIAS...

```
class Estudante {
    public Estudante( String nome, char sexo,
        int matricula) {
        this.sexo = sexo;
        this.nome = nome;
        this.matricula = matricula;
    } ...
}

class EstudanteTutor extends Estudante {
    public EstudanteTutor( String nome, char sexo,
        int matricula, double bolsa, String disciplina){
        super( nome, sexo, matricula );
        this.bolsa = bolsa;
        this.disciplina = disciplina;
    }
}
```

Prof. Marcos Vinicius – UFC/Russas - POO

29/37



CUIDADO!!!

Como a **chamada this()** também tem que ser a **primeira sentença dentro de um construtor**, então o **uso das duas chamadas é exclusiva**, ou seja, somente uma das duas pode ser usada de cada vez!

ENCADEAMENTO DE CONSTRUTORES

- Se a classe não definir **nenhum** construtor, seu construtor **default** será chamado quando da criação de objetos.
- Se não for especificada uma chamada **this()** ou **super()** dentro do construtor, a JVM **inclui implicitamente** uma chamada ao construtor **default** da **superclasse**.
- Nesse caso, se a superclasse imediata **não** possuir construtor **default**, ocorrerá **erro de compilação**.

A JVM é metida!



Prof. Marcos Vinicius – UFC/Russas - POO

31/37

TROCANDO EM MIÚDOS...

```
public class A
{
    public A()
    {}
}

class B extends A
{
    //sem construtores
    //...
}
```

Equivale a

```
public class A
{
    public A() {
        super();
    }
}

class B extends A
{
    public B() {
        super();
    }
    // ...
}
```

Prof. Marcos Vinicius – UFC/Russas - POO

32/37

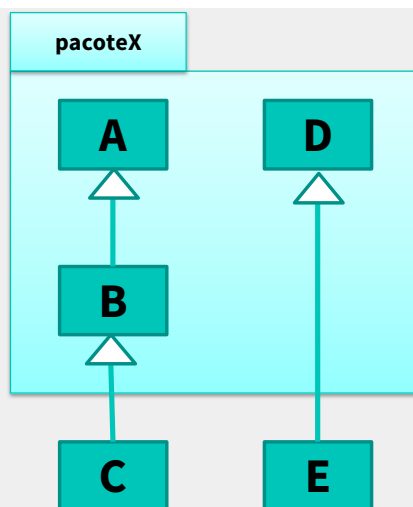


Visibilidade em Java

SE LIGA!

Variável ou método na classe "A"	Pode ser acessado por
+ Público	A, B, C, D, E
# Protegido	A, B, C, D
Pacote	A, B, D
- Privado	A

Maior Encapsulamento



CUIDADO COM A VOADORA...

Classe só tem
visibilidade
pública ou de
pacote!



Prof. Marcos Vinicius – UFC/Russas - POO

35/37



Vamos
brincar um
pouco no
AME?



Obrigado!

Mais alguma dúvida?

Acesse o **AME** para mais informações e treinamento do **NERDS!**

<http://ame2.russas.ufc.br>

