

Programação Orientada a Objetos com a Linguagem JAVA

Fábio Pereira Botelho

M.Sc. Ciência da Computação UFPE
botelho.fabio@hotmail.com

Anápolis – GO, 2024/02

Objetivo

1. Discutir os conceitos de Programação Orientada a Objetos Confrontando com a Clássica Programação Estruturada
 2. Apresentar a UML como instrumento básico para a modelagem de softwares orientados a Objeto
 3. Apresentar a linguagem JAVA como referencial fiel aos conceitos de OO
-
-

Modelo de Orientação a Objetos (OO)

- ✓ O modelo de Objetos foi introduzido inicialmente como modelo de programação (Simula-67 e Smalltalk-80)
 - ✓ Foi estendido para Bancos de Dados e Representação do Conhecimento, além de modelagem de software
-
-

Paradigma OO X Estruturado

- ✓ As linguagens procedurais como Pascal e C representam conceitos complexos do mundo real como estruturas de dados e funções como operações que podem ser efetuadas nos dados.
 - ✓ No modelo de objetos as coisas são diferentes pois a estrutura de dados (propriedades) e as operações (métodos) são reunidas em uma única construção denominada **classe**
-
-

Paradigma OO X Estruturado

Linguagem C (procedural)

```
Typedef struct {  
    char nome[50];  
    int numeroRua;  
    char rua[100];  
    char cidade[50];  
    int CEP;  
    char estado[2];  
} Endereco;  
  
Endereco *criarEndereco(char *n, int nr, char *r, char *c, int  
cep, char *e);
```

Linguagem JAVA (OO)

```
public class Endereco {  
    String nome, rua, cidade, estado;  
    int numeroRua, CEP;  
    Endereco(String n, String r, String c, String e, int nr,  
    int cep) {  
        nome = n;  
        rua = r;  
        cidade = c;  
        estado = e;  
        numeroRua = nr;  
        CEP = cep;  
    }  
}
```

Paradigma OO X Estruturado

- ✓ O modelo procedural guarda funções e atributos (propriedades) em separado, enquanto que o OO os guarda juntos.
 - ✓ No modelo procedural, quando se usa uma função, é preciso saber como ela age sobre os parâmetros. No OO pode-se imaginar que os próprios objetos se modificam.
-
-

Paradigma OO X Estruturado

- ✓ A classe possui a vantagem da proximidade física dos dados e das funções que pertencem ao mesmo conceito do mundo real, o que torna mais fácil modificar a implementação (e.g. acrescentar um atributo)
 - ✓ No modelo de objetos, a classe é um tipo que pode ser usada para criar objetos (instâncias). É possível nas instâncias, usar os métodos definidos para a classe
-
-

Definições Paradigma OO

- ✓ Classe
 - ✓ Objeto
 - ✓ Instância
 - ✓ Propriedade
 - ✓ Atributo
 - ✓ Método
 - ✓ Encapsulamento
 - ✓ Operação
 - ✓ Mensagem
 - ✓ Herança
 - ✓ Hierarquia de Herança
 - ✓ Polimorfismo
 - ✓ Interface
-
-

Definições Paradigma OO

- ✓ **Classe**

- ✓ É a modelagem de um conceito do mundo Real. As propriedades associadas ao conceito são denominadas **Atributos** (variáveis) e **Operações** (funções).

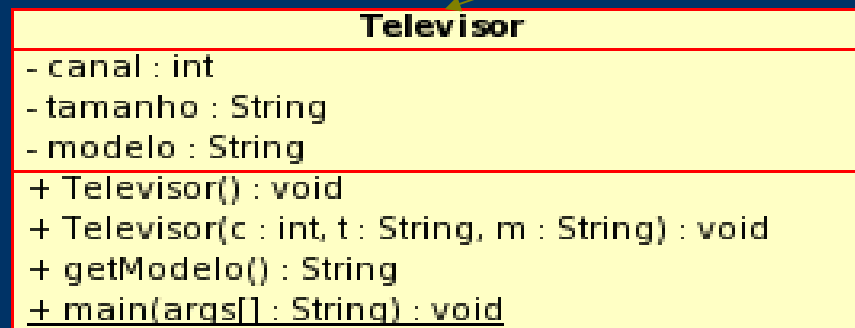
- ✓ Descreve os atributos que seus objetos irão ter e os métodos que irão executar.



Definições Paradigma OO

✓ Classe

Definição UML da classe Televisor



Definições Paradigma OO

✓ Classe JAVA

```
public class Televisor {  
    private int canal;  
    private String tamanho;  
    private String modelo;  
    Televisor() { // método construtor  
        canal = 4;  
        tamanho = "15 polegadas";  
        modelo = "SEMP TOSHIBA";  
    }  
    Televisor(int c,String t,String m) { // método construtor - sobrecarga  
        canal = c;  
        tamanho = t;  
        modelo = m;  
    }  
    String getModelo() {  
        return modelo;  
    }  
}
```

```
public static void main ( String args[] ) {  
    Televisor tv1 = new Televisor();  
    Televisor tv2 = new Televisor(10,new String("10 polegadas"),new  
String("Gradiente"));  
    System.out.println("O objeto tv1 é " + tv1.getModelo());  
    System.out.println("O objeto tv2 é " + tv2.getModelo());  
    }  
}
```

O resultado da execução é:

```
[root@fabio uml-generated-code]# java Televisor  
O objeto tv1 é SEMP TOSHIBA  
O objeto tv2 é Gradiente
```

Definições Paradigma OO

✓ Objeto

- ✓ Cada objeto do mundo real pode ser representado por um objeto. E.g.: Cada professor, cada aula no domínio de uma universidade.
 - ✓ A estrutura do objeto como as operações que ele pode executar são descritas pela classe.
 - ✓ Dois objetos com exatamente os mesmos atributos são dois objetos distintos. Ocupam espaços de memória diferentes quando instanciados durante a execução do programa.
-
-

Definições Paradigma OO

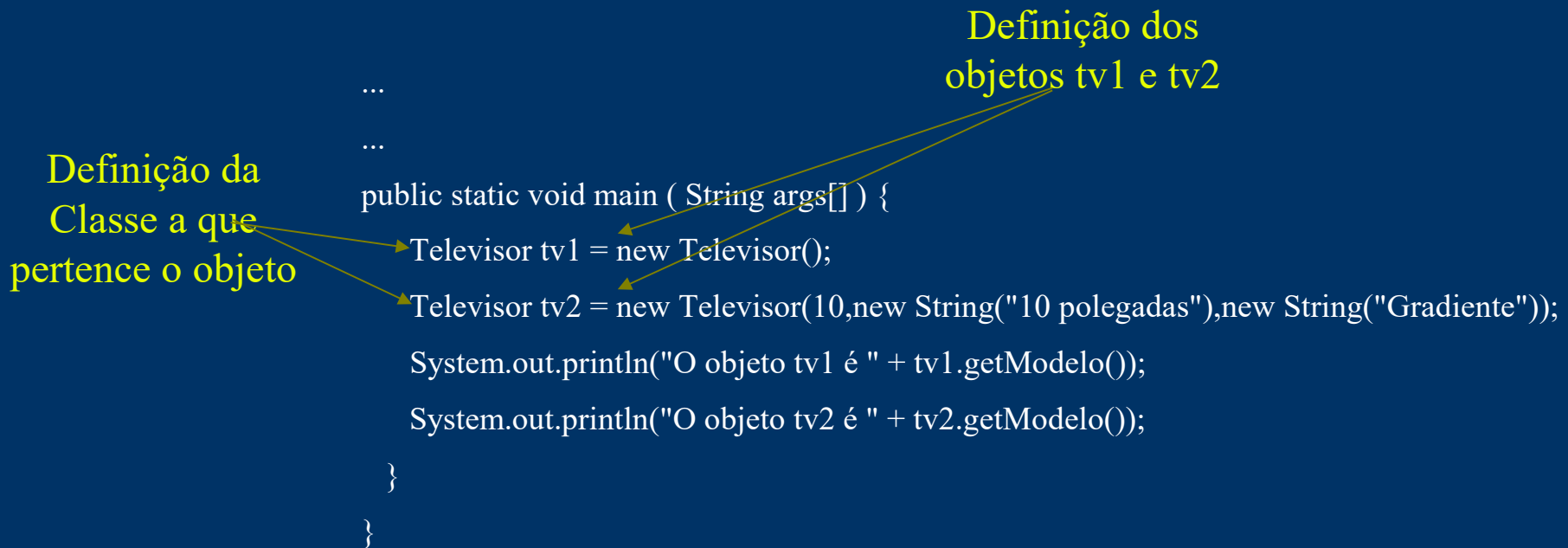
✓ Objeto

✓ Dado o exemplo da classe Televisor já definida, teríamos os exemplos dos objetos tv1 e tv2.

Definição da Classe a que pertence o objeto

Definição dos objetos tv1 e tv2

```
...  
...  
public static void main ( String args[] ) {  
    Televisor tv1 = new Televisor();  
    Televisor tv2 = new Televisor(10,new String("10 polegadas"),new String("Gradiente"));  
    System.out.println("O objeto tv1 é " + tv1.getModelo());  
    System.out.println("O objeto tv2 é " + tv2.getModelo());  
}  
}
```



Definições Paradigma OO

✓ Instância

✓ Entende-se como a aplicação concreta de um dado conceito abstrato. Um objeto é uma instância de uma classe. No exemplo dado, os objetos tv1 e tv2 são instâncias da classe Televisor.

```
...  
...  
public static void main ( String args[] ) {  
    Televisor tv1 = new Televisor();  
    Televisor tv2 = new Televisor(10,new String("10 polegadas"),new String("Gradiente"));  
    System.out.println("O objeto tv1 é " + tv1.getModelo());  
    System.out.println("O objeto tv2 é " + tv2.getModelo());  
}  
}
```

Os objetos tv1 e tv2 são instanciados através de diferentes métodos construtores da classe Televisor (sobrecarga)

Definições Paradigma OO

✓ Propriedade

✓ Um conceito é definido por um conjunto de propriedades (e.g. Uma pedra é dura, inanimada, sólida). Uma classe modela um conceito definindo algumas de suas propriedades. Propriedades podem ser relevantes ou não para o conceito que se quer modelar.

Canal, tamanho e modelo são
propriedades relevantes de um
Televisor.

Em nosso modelo, a propriedade peso
não seria relevante

```
public class Televisor {  
    int canal;  
    String tamanho;  
    String modelo;  
    ...  
    ...  
}
```

Definições Paradigma OO

✓ Atributo

✓ Pode ser entendido como uma propriedade relevante (importante) para a modelagem de uma dada classe (e.g. A data de nascimento de um aluno)

Canal, tamanho e modelo são atributos identificados entre as propriedades de um Televisor

Em nosso modelo, a propriedade peso não seria relevante, logo não seria definida como um atributo.

```
public class Televisor {  
    int canal;  
    String tamanho;  
    String modelo;  
    ...  
    ...  
}
```

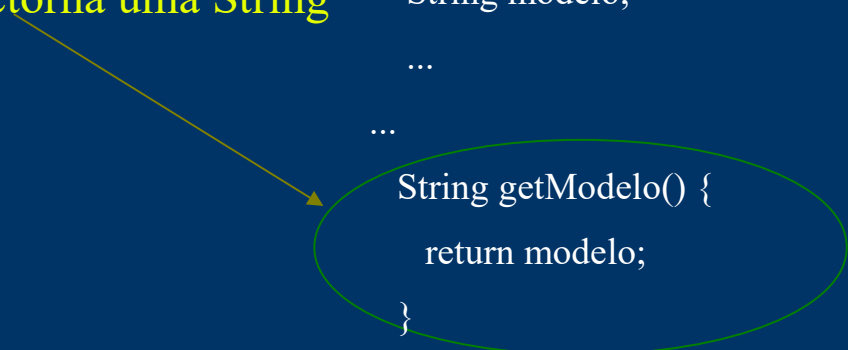

Definições Paradigma OO

✓ Método

✓ Uma propriedade importante de uma classe que pode ser representada como uma função (e.g. A idade de um dado aluno pode ser calculada a partir da data de nascimento e da data corrente)

A classe Televisor, define o método getModelo() que retorna uma String

```
public class Televisor {  
    int canal;  
    String tamanho;  
    String modelo;  
    ...  
    ...  
    String getModelo() {  
        return modelo;  
    }  
    ...  
}
```



Definições Paradigma OO

✓ **Operação**

- ✓ Entende-se como um método mais abstrato. O método se refere implicitamente a uma função. A operação é mais abstrata, referindo-se à idéia do que a função faz, sem se preocupar com os detalhes da sua implementação.
- ✓ O método implementa uma operação



Definições Paradigma OO

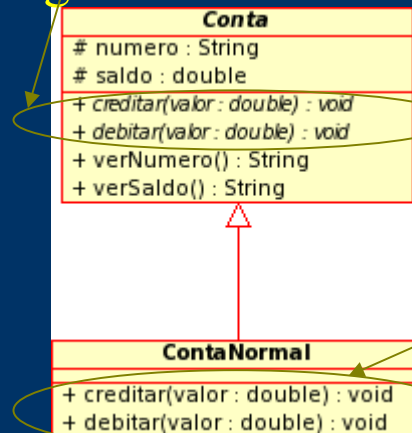
✓ Operação

A Classe Abstrata Conta define as operações (abstract) creditar e debitar sem se preocupar como serão implementadas

```
abstract class Conta {  
    protected String numero;  
    protected double saldo;  
    abstract void creditar(double valor);  
    abstract void debitar(double valor);  
    String verNumero() {  
        return numero;  
    }  
    double verSaldo() {  
        return saldo;  
    }  
}
```

A Classe Abstrata Conta também implementa os métodos verNumero e verSaldo

Diagrama de Classes UML



```
class ContaNormal extends Conta {  
    void creditar(double valor) {  
        saldo = saldo + valor;  
    }  
    void debitar(double valor) {  
        saldo = saldo - valor;  
    }  
}
```

A Classe ContaNormal que herda da Classe Conta implementa os métodos creditar e debitar definidos como operações creditar e debitar na Classe Pai Conta (**Polimorfismo**)

Definições Paradigma OO

✓ Mensagem

- ✓ Manda-se uma **mensagem** a um **objeto** a fim de pedir a ele para executar uma operação particular.
- ✓ Do exemplo da classe Televisor, temos:

```
...  
...  
public static void main ( String args[] ) {  
    Televisor tv1 = new Televisor();  
    Televisor tv2 = new Televisor(10,new String("10 polegadas"),new String("Gradiente"));  
    System.out.println("O objeto tv1 é " + tv1.getModelo());  
    System.out.println("O objeto tv2 é " + tv2.getModelo());  
}  
  
}
```

Imprime na tela a String “O objeto tv? É” , seguido do resultado da chamada do método getModelo() em cada objeto, i.e tv1 e tv2

Mensagens enviadas aos objetos tv1 e tv2 a fim de que eles executem, cada um, o método getModelo() que retorna uma String contendo o modelo de cada objeto instanciado.

Definições Paradigma OO

✓ Herança

- ✓ É a modelagem da noção de especialização / generalização.
- ✓ No mundo real, conceitos são especializados uns dos outros.
- ✓ e.g. Uma Conta Especial é uma especialização de uma Conta Bancária; Um professor é uma especialização de funcionário em uma instituição de ensino

Definições Paradigma OO

- ✓ Herança

- ✓ Os conceitos mais especializados possuem todas as propriedades dos mais generalizados somados às suas próprias propriedades.



Definições Paradigma OO

✓ Herança

- ✓ No modelo de objetos, uma classe filha possui todos os atributos e métodos da classe pai.
 - ✓ A classe filha pode acrescentar atributos e métodos
 - ✓ A classe filha pode redefinir alguns métodos da classe pai (sobrecarga de método)
 - ✓ A classe filha não pode tirar propriedades definidas na classe pai
-
-

Definições Paradigma OO

✓ Herança

✓ Para exemplificar Herança, vamos criar as classes TelevisorComDVD e CR, além de remodelar a classe Televisor.

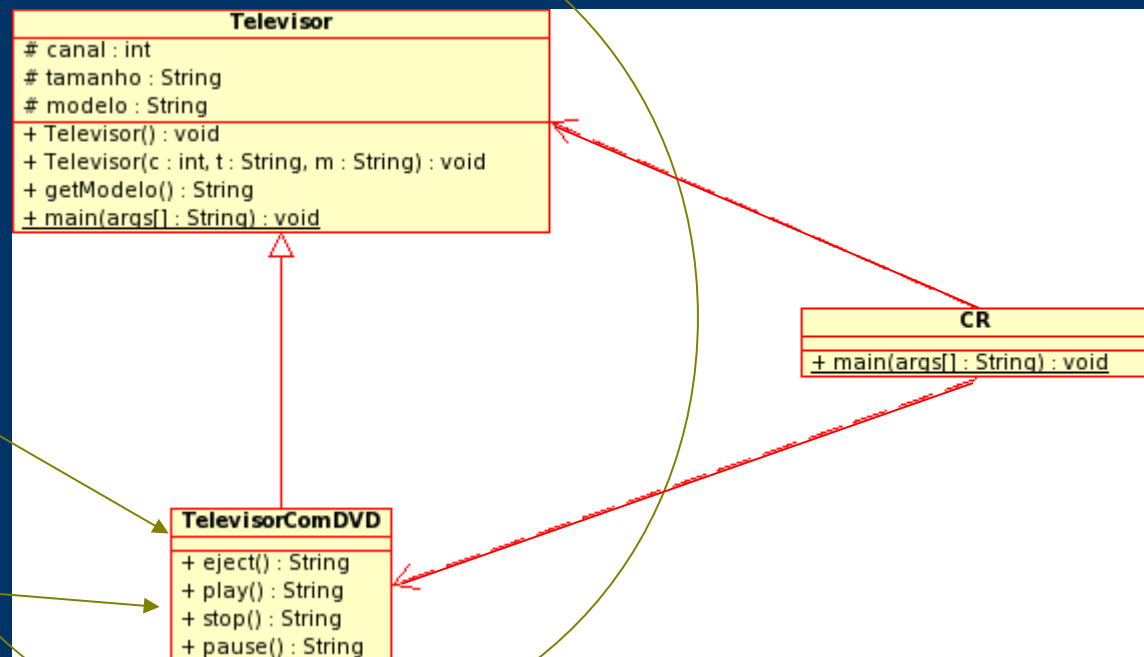
Modelagem de herança com UML (Diagrama de Classes)

Classe

TelevisorComDVD herda atributos e métodos da classe Televisor

Classe

TelevisorComDVD acrescenta novos métodos



Definições Paradigma OO

Classe

✓ Herança

TelevisorComDVD herda
atributos e métodos da
classe Televisor



```
graph TD; A[TelevisorComDVD herda atributos e métodos da classe Televisor] --> B[public class TelevisorComDVD extends Televisor{...}];
```

```
public class Televisor {  
    protected int canal;  
    protected String tamanho;  
    protected String modelo;  
    public Televisor ( ) {  
        canal = 4;  
        tamanho = "15 polegadas";  
        modelo = "SEMP TOSHIBA";  
    }  
    public Televisor ( int c, String t, String m) {  
        canal = c;  
        tamanho = t;  
        modelo = m;  
    }  
    public String getModelo ( ) {  
        return modelo;  
    }  
}
```

```
public class TelevisorComDVD extends Televisor {  
    TelevisorComDVD ( ) {  
        canal = 31;  
        tamanho = "20 polegadas";  
        modelo = "PHILCO";  
    }  
    public String eject ( ) {  
        return "Eject ativado";  
    }  
    public String play ( ) {  
        return "Função Play ativada";  
    }  
    public String stop ( ) {  
        return "Função Stop ativada";  
    }  
    public String pause ( ) {  
        return "Função Pause ativada";  
    }  
}
```

Definições Paradigma OO

✓ Herança

Método definido na classe Pai
Televisor e disponível na classe
Filha TelevisorComDVD

```
public class CR {  
    public static void main ( String args[]) {  
        Televisor tv1 = new Televisor();  
        Televisor tv2 = new Televisor(10,new String("10 polegadas"),new String("Gradiente"));  
        TelevisorComDVD tvdvd1 = new TelevisorComDVD();  
        System.out.println("O objeto tv1 é " + tv1.getModelo());  
        System.out.println("O objeto tv2 é " + tv2.getModelo());  
        System.out.println("O objeto tvdvd1 é " + tvdvd1.getModelo());  
        System.out.println("    " + tvdvd1.eject());  
        System.out.println("    " + tvdvd1.stop());  
        System.out.println("    " + tvdvd1.play());  
        System.out.println("    " + tvdvd1.pause());  
    }  
}
```

Resultado da Execução:

```
[root@fabio uml-generated-code]# java CR  
O objeto tv1 é SEMP TOSHIBA  
O objeto tv2 é Gradiente  
O objeto tvdvd1 é PHILCO  
    Eject ativado  
    Função Stop ativada  
    Função Play ativada  
    Função Pause ativada
```

Definições Paradigma OO

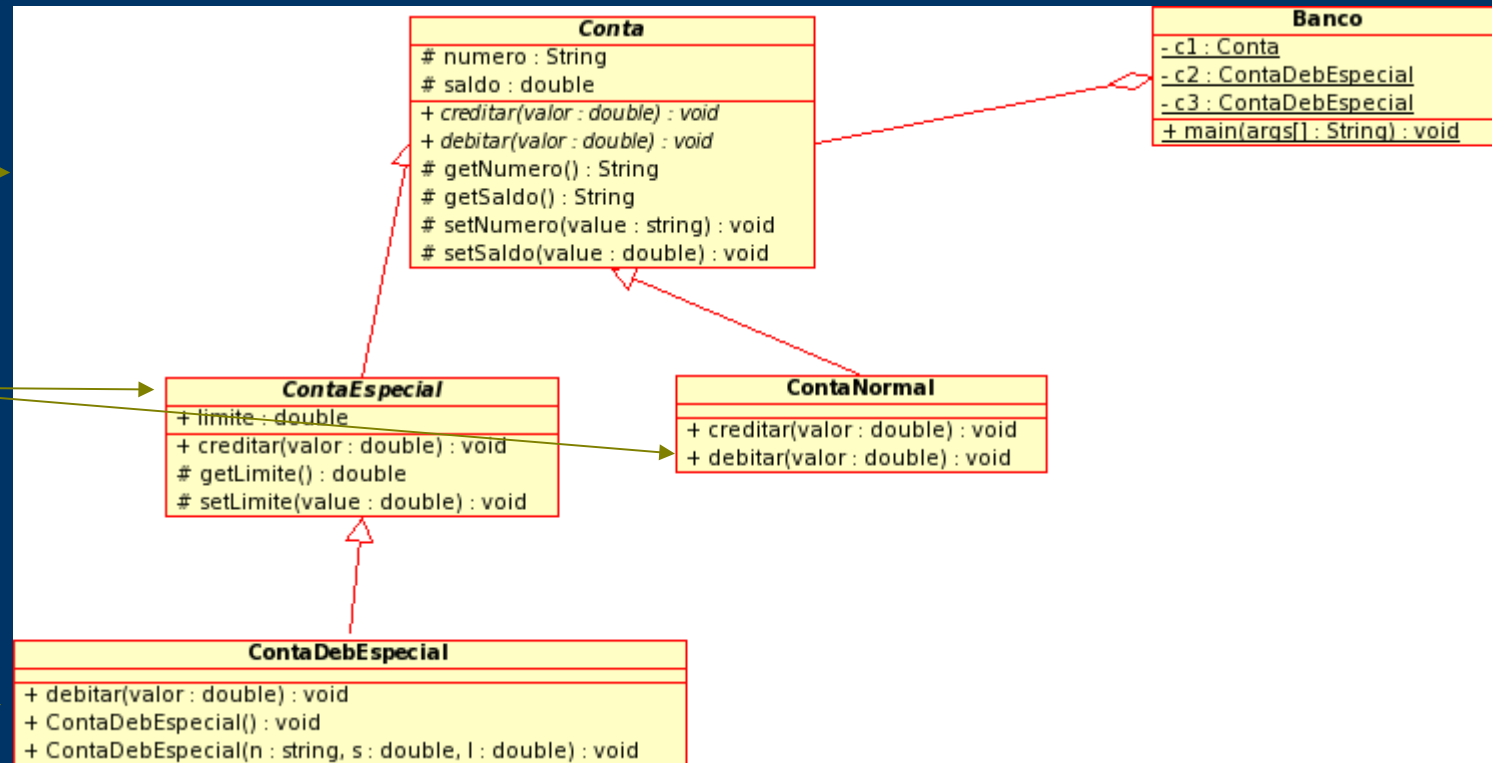
✓ Hierarquia de Herança

✓ Todas as relações de herança entre todas as classes formam uma árvore que se chama hierarquia de herança. No domínio do sistema bancário teríamos:

Conforme modelado em UML (Diagrama de Classes)

ContaEspecial e ContaNormal herdam da classe Pai Conta

ContaDebEspecial herda da Classe Pai ContaEspecial



Definições Paradigma OO

- ✓ Hierarquia de Herança
 - ✓ Exemplo Banco implementado em JAVA
 - ✓ Conta.java
 - ✓ ContaNormal.java
 - ✓ ContaEspecial.java
 - ✓ ContaDebEspecial.java
 - ✓ Banco.java
-
-

Definições Paradigma OO

✓ Hierarquia de Herança

✓ Conta.java

```
abstract public class Conta {  
    protected String numero;  
    protected double saldo;  
    protected String getNumero ( ) {  
        return numero;  
    }  
  
    protected void setNumero ( String value ) {  
        numero = value;  
    }  
  
    protected double getSaldo ( ) {  
        return saldo;  
    }  
  
    protected void setSaldo ( double value ) {  
        saldo = value;  
    }  
  
    abstract void creditar (double valor);  
    abstract void debitar (double valor);  
}
```

✓ ContaNormal.java

```
public class ContaNormal extends Conta {  
  
    public void creditar (double valor) {  
        saldo = saldo + valor;  
    }  
  
    public void debitar (double valor) {  
        if ((saldo-valor) >= 0) {  
            saldo = saldo - valor;  
        }  
    }  
}
```

Conta Normal não possui limite, logo em uma operação de débito apenas o saldo está disponível.

Definições Paradigma OO

✓ Hierarquia de Herança

✓ ContaEspecial.java

```
abstract public class ContaEspecial extends Conta {  
  
    protected double limite;  
  
    protected void setLimite ( double value ) {  
        limite = value;  
    }  
  
    protected double getLimite () {  
        return limite;  
    }  
  
    public void creditar ( double valor) {  
        saldo = saldo + valor;  
    }  
}
```

✓ ContaDebEspecial.java

```
public class ContaDebEspecial extends ContaEspecial {  
  
    ContaDebEspecial() {  
        super();  
    }  
  
    ContaDebEspecial(String n,double s, double l){  
        numero = n;  
        saldo = s;  
        limite = l;  
    }  
  
    void debitar (double valor) {  
        if ((limite + saldo - valor) >= 0) {  
            saldo = saldo - valor;  
        }  
    }  
}
```

Definições Paradigma OO

✓ Hierarquia de Herança

✓ Banco.java

```
class Banco {  
  
    public static void main (String args[]) {  
        Conta c1;  
        c1 = new ContaNormal();  
        c1.setNumero(new String("1654-3"));  
        c1.setSaldo(500);  
        ContaDebEspecial c2 = new ContaDebEspecial();  
        ContaDebEspecial c3 = new ContaDebEspecial(new String("4067-6"),2500,5050);  
        c2.setNumero(new String("4067-6"));  
        c2.setSaldo(2500);  
        c2.setLimite(1000.67);  
        System.out.println("A conta número " + c1.getNumero() + " possui saldo " + c1.getSaldo());  
        c1.creditar(1000);  
        System.out.println("Após o credito de R$ 1000,00, a conta número " + c1.getNumero() + " passou a ter saldo " + c1.getSaldo());  
        c1.debitar(100);  
        System.out.println("Após o débito de R$ 100,00, a conta número " + c1.getNumero() + " passou a ter saldo " + c1.getSaldo());  
        System.out.println("");  
        System.out.println("A conta número " + c2.getNumero() + " possui saldo " + c2.getSaldo());  
        c2.debitar(500);  
        System.out.println("A conta número " + c2.getNumero() + " possui saldo " + c2.getSaldo() + " Após débito de R$ 500,00, a conta número " + c2.getNumero() + " passou a ter saldo " + c2.getSaldo());  
        System.out.println("A conta número " + c2.getNumero() + " possui saldo " + c2.getSaldo() + " e Limite de " + c2.getLimite());  
        c2.setLimite(10000);  
        System.out.println("A conta número " + c2.getNumero() + " possui saldo " + c2.getSaldo() + " e novo Limite de " + c2.getLimite());  
    }  
}
```

Definições Paradigma OO

- ✓ Hierarquia de Herança
- ✓ Compilação e Execução
 - `javac *.java`
 - `java Banco`

```
[root@fabio Banco]# java Banco
```

A conta número 1654-3 possui saldo 500.0

Após o credito de R\$ 1000,00, a conta número 1654-3 passou a ter saldo 1500.0

Após o débito de R\$ 100,00, a conta número 1654-3 passou a ter saldo 1400.0

A conta número 4067-6 possui saldo 2500.0

A conta número 4067-6 possui saldo 2000.0 Após débito de R\$ 500

A conta número 4067-6 possui saldo 2000.0 e Limite de 1000.67

A conta número 4067-6 possui saldo 2000.0 e novo Limite de 10000.0

Definições Paradigma OO

- ✓ **Polimorfismo**

- ✓ O Polimorfismo é um termo utilizado para descrever a situação em que um nome pode referir-se a diferentes métodos. Em java existem dois tipos de polimorfismo: **tipo sobrecarga e tipo sobreposição**

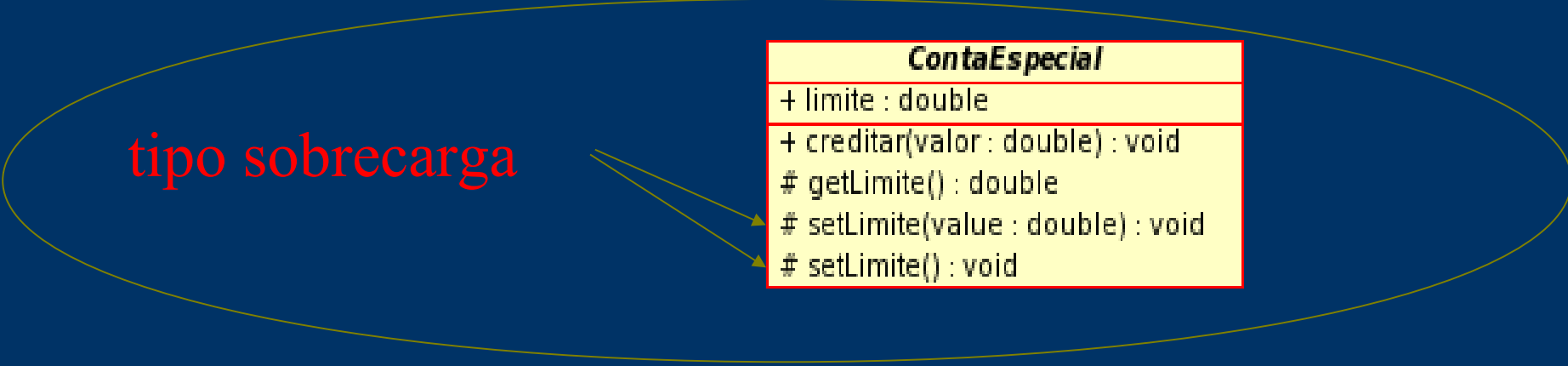


Definições Paradigma OO

- ✓ Polimorfismo
- ✓ O Polimorfismo tipo sobrecarga é o que estudamos como sobrecarga de método, ou seja, temos o mesmo nome de método em uma única classe.

Extensão do exemplo da Classe Abstrata ContaEspecial do Banco exemplo dado

tipo sobrecarga



ContaEspecial
+ limite : double
+ creditar(valor : double) : void
getLimite() : double
setLimite(value : double) : void
setLimite() : void

Definições Paradigma OO

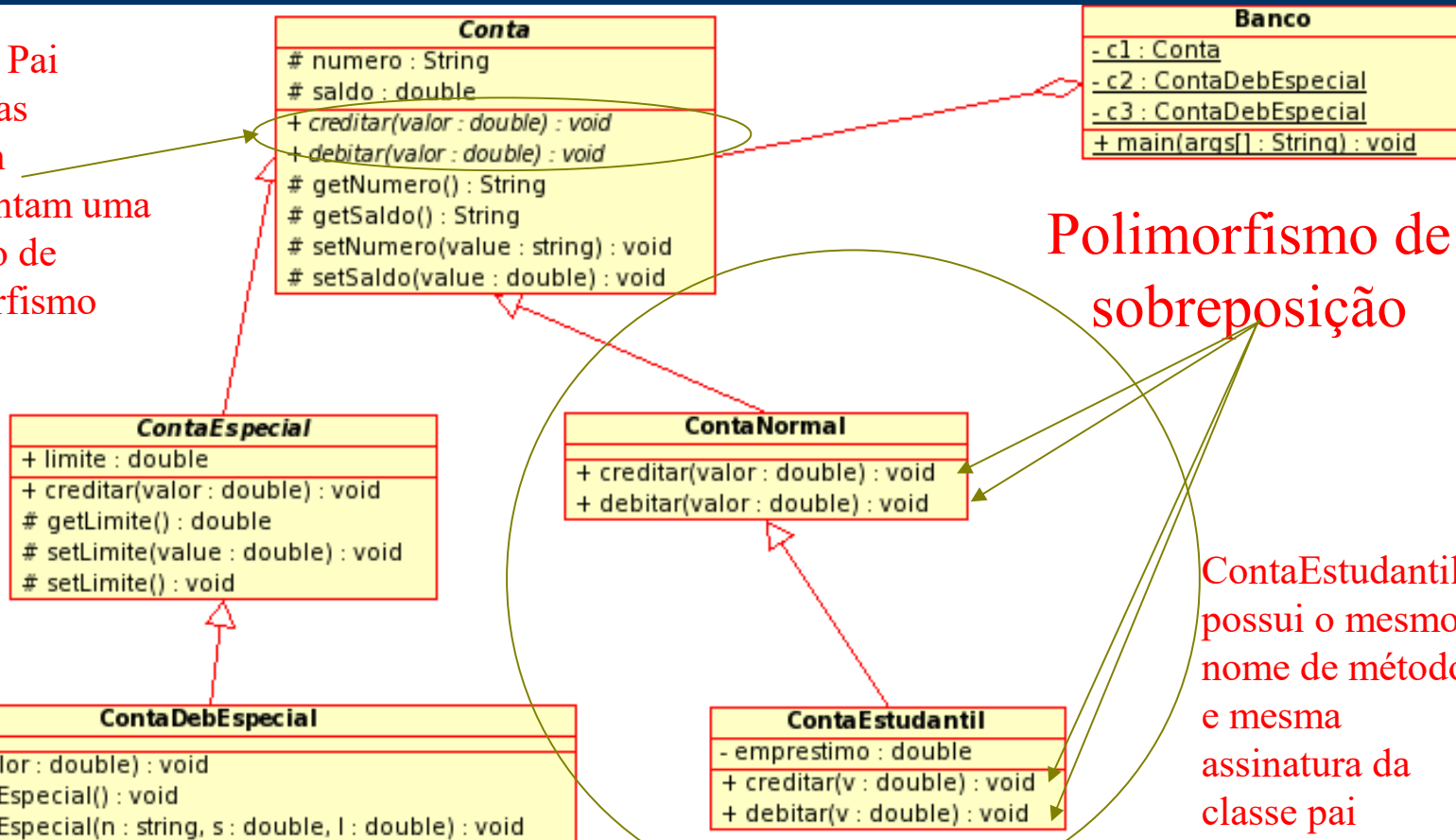
- ✓ Polimorfismo
- ✓ O Polimorfismo tipo sobreposição ocorre quando um método da classe filha tem o mesmo nome de método e a mesma assinatura do método definido na classe pai (super-classe)

Definições Paradigma OO

✓ Polimorfismo

✓ Polimorfismo de Sobreposição

Classes Pai
Abstratas
também
representam uma
situação de
Polimorfismo

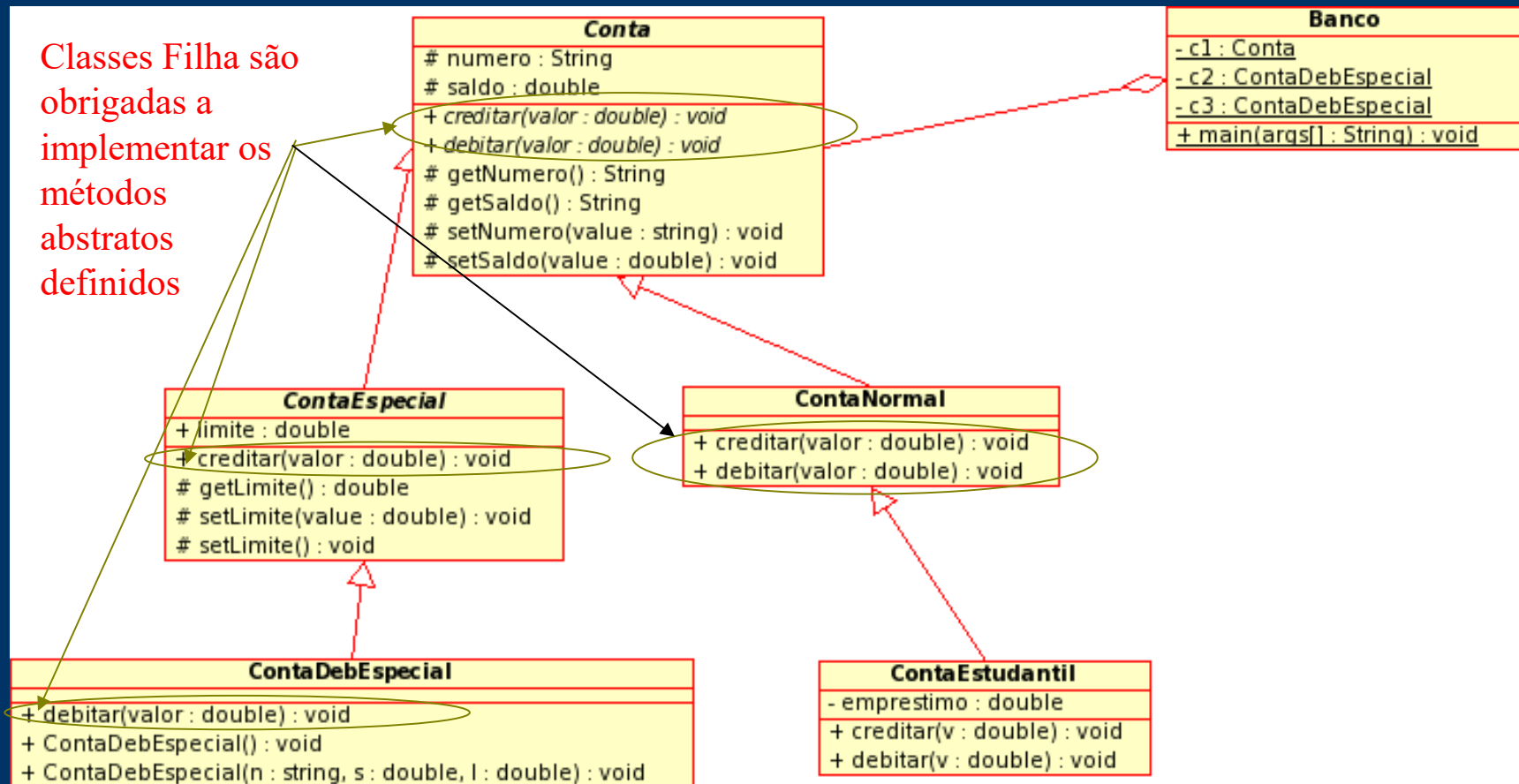


Definições Paradigma OO

✓ Polimorfismo

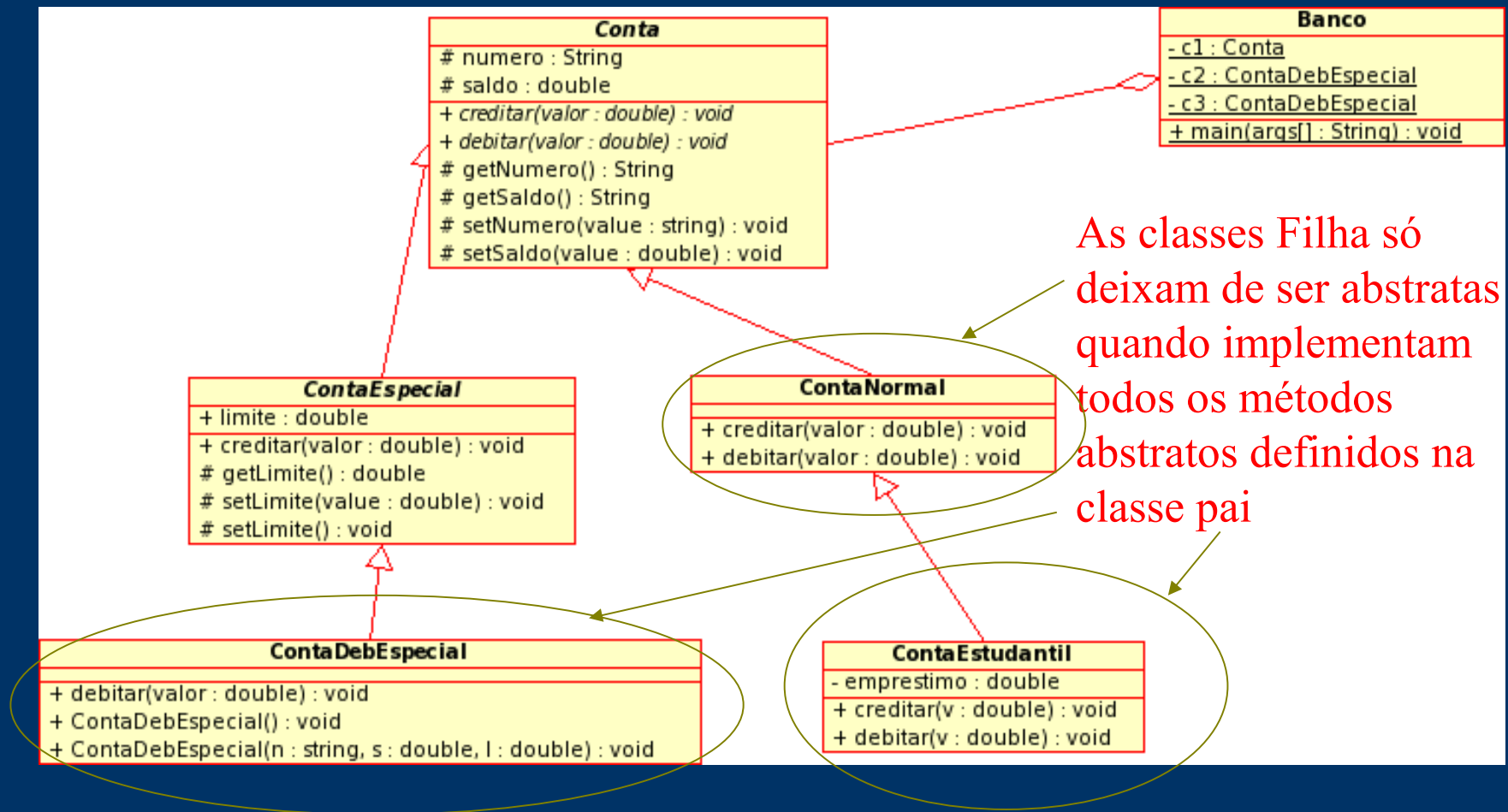
✓ Polimorfismo de Sobreposição

Classes Filha são obrigadas a implementar os métodos abstratos definidos



Definições Paradigma OO

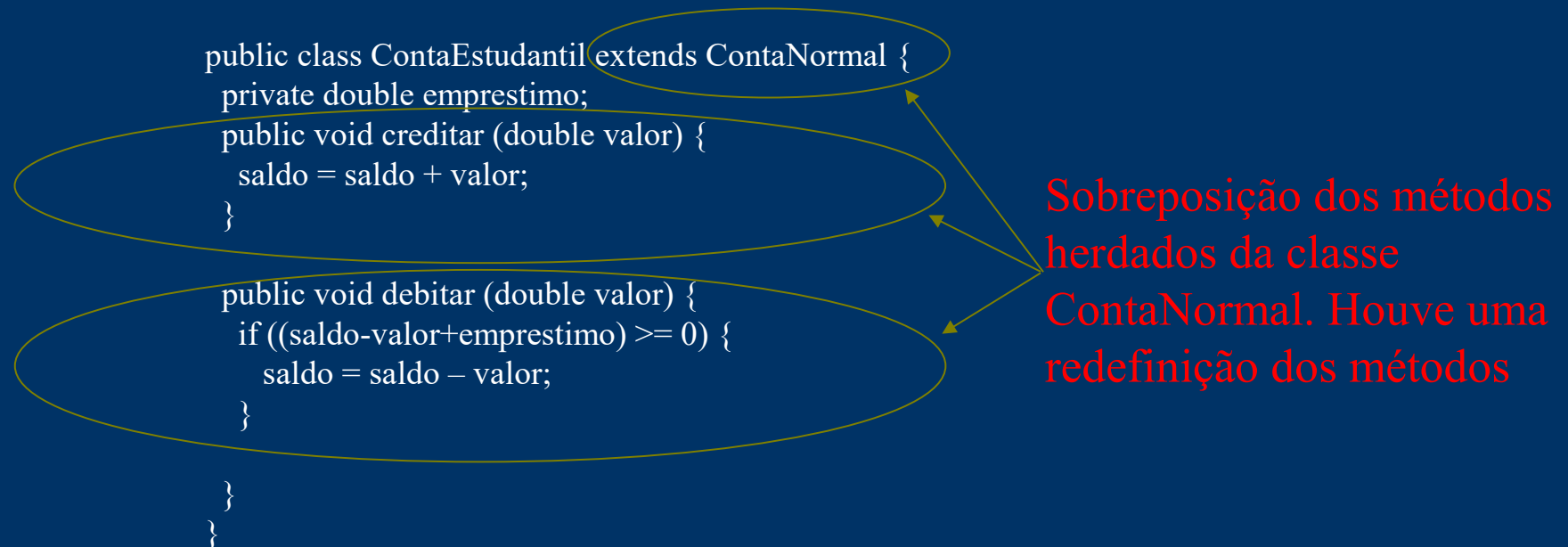
- ✓ Polimorfismo
- ✓ Polimorfismo de Sobreposição



Definições Paradigma OO

- ✓ Polimorfismo
- ✓ Polimorfismo de Sobreposição
- ✓ Exercício

Implemente em JAVA a Classe ContaEstudantil como vista no diagrama de classes UML anteriormente



Definições Paradigma OO

- ✓ Interface
- ✓ Permite proteger o exterior da classe com relação às mudanças ocorridas no interior da classe
- ✓ Fornece os nomes dos métodos, sem as respectivas implementações
- ✓ Diz-se que uma classe implementa uma interface.
- ✓ Uma classe pode implementar várias interfaces (herança múltipla não disponível em JAVA)



Definições Paradigma OO

- ✓ Interface
- ✓ Todos os métodos de uma interface são implicitamente públicos (palavra reservada public) e abstratos (abstract). Não existem outros tipos permitidos
- ✓ Todas as variáveis em uma interface são implicitamente public, final e static. Não existem outras possibilidades



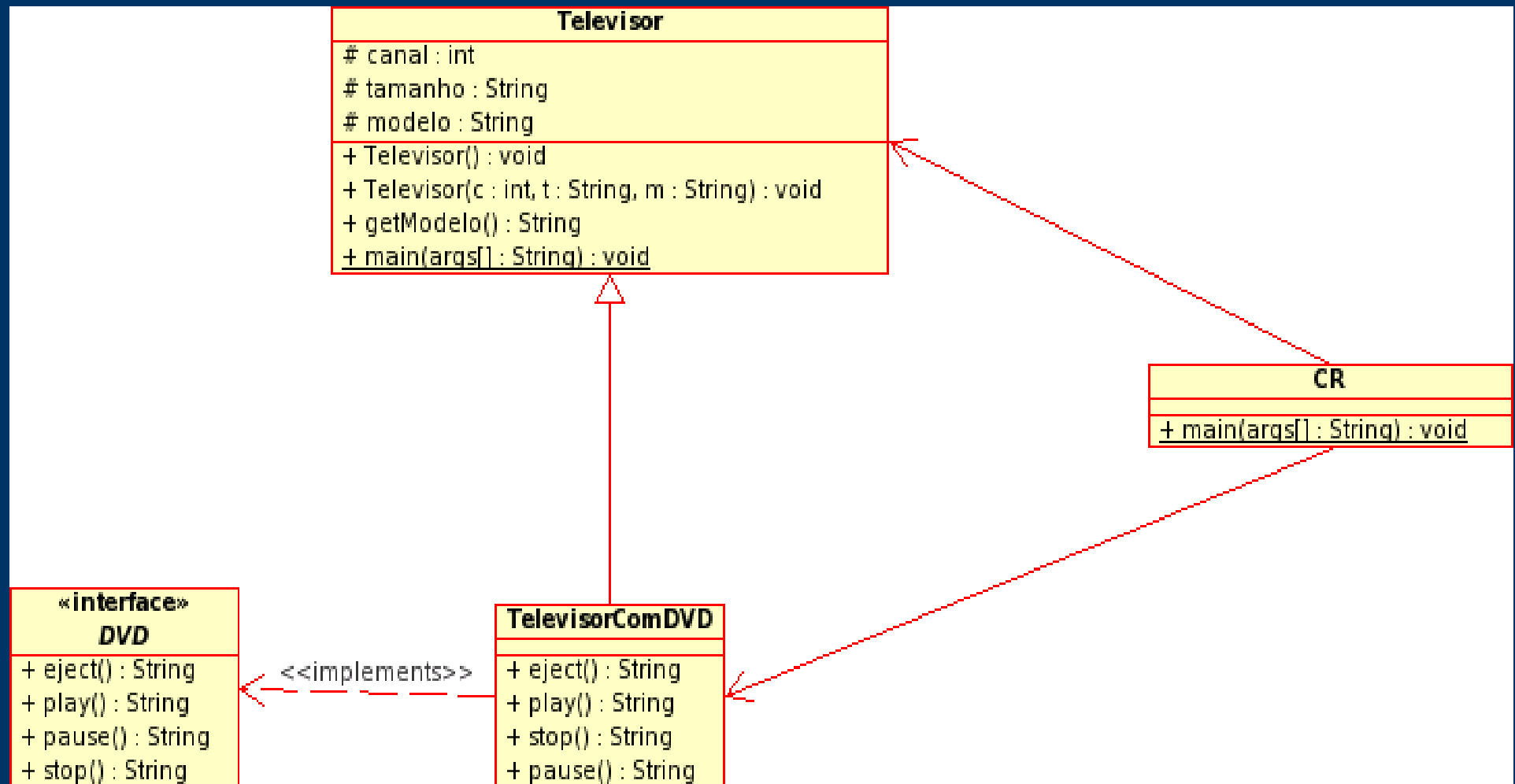
Definições Paradigma OO

- ✓ Interface
- ✓ Todas as operações definidas na Interface devem ser implementados pela classe que usa a interface
- ✓ Uma classe interface não tem uma raiz progenitora que conduza a um objeto, como é o caso da classe abstrata
- ✓ As interfaces encontram-se em uma hierarquia independente, podendo ser implementada em qualquer parte da árvore de classes



Definições Paradigma OO

- ✓ Interface
- ✓ Representação UML (Diagrama de Classes)



Definições Paradigma OO

- ✓ Interface
- ✓ Codificação em JAVA (Alteração no Projeto Televisor)

Interface DVD

```
public interface DVD {  
    public String eject();  
    public String play();  
    public String stop();  
    public String pause();  
}
```

TelevisorComDVD
implementa todas as
operações definidas na
Interface DVD

Classe TelevisorComDVD

```
public class TelevisorComDVD extends Televisor implements DVD {  
    TelevisorComDVD () {  
        canal = 31;  
        tamanho = "20 polegadas";  
        modelo = "PHILCO";  
    }  
    public String eject () {  
        return "Eject ativado";  
    }  
    public String play () {  
        return "Função Play ativada";  
    }  
    public String stop () {  
        return "Função Stop ativada";  
    }  
    public String pause () {  
        return "Função Pause ativada";  
    }  
}
```

Como se fosse
Herança Múltipla

Definições Paradigma OO

- ✓ Interface
- ✓ Codificação em JAVA (Alteração no Projeto Televisor)
- ✓ Exercício:
 - ➔ Crie um Projeto no Eclipse denominado Televisor e acrescente a interface DVD conforme demonstrado no slide anterior.
 - ➔ Execute a classe CR (possui o método main)

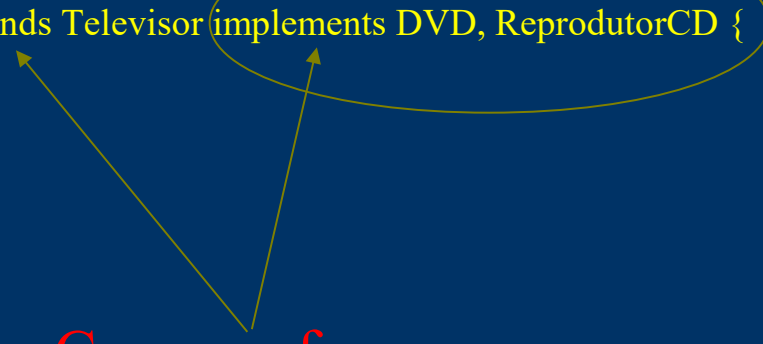


Definições Paradigma OO

- ✓ Interface
- ✓ Se existisse mais de uma interface definida, e.g. ReprodutorCD, TelevisorComDVD poderia implementar várias interfaces

Classe TelevisorComDVD

```
public class TelevisorComDVD extends Televisor implements DVD, ReprodutorCD {  
    TelevisorComDVD () {  
        canal = 31;  
        tamanho = "20 polegadas";  
        modelo = "PHILCO";  
    }  
    public String eject () {  
        return "Eject ativado";  
    }  
    public String play () {  
        return "Função Play ativada";  
    }  
    public String stop () {  
        return "Função Stop ativada";  
    }  
    public String pause () {  
        return "Função Pause ativada";  
    }  
}
```



Como se fosse
Herança Múltipla

Definições Paradigma OO

- ✓ Encapsulamento
- ✓ Entende-se como a combinação de dados e métodos em uma estrutura de dados simples
- ✓ Agrupa todos os componentes de um objeto
- ✓ Define como os programas irão referenciar os dados de um objeto
- ✓ Programas somente acessam dados (i.e. atributos ou variáveis) privados através de métodos declarados como públicos



Definições Paradigma OO

- ✓ Encapsulamento (moderadores de acesso)
- ✓ São empregados para restringir o acesso a um método ou a um atributo
- ✓ Independentemente do moderador escolhido, um método ou um atributo podem ser acessíveis a partir de qualquer outro método contido na mesma classe

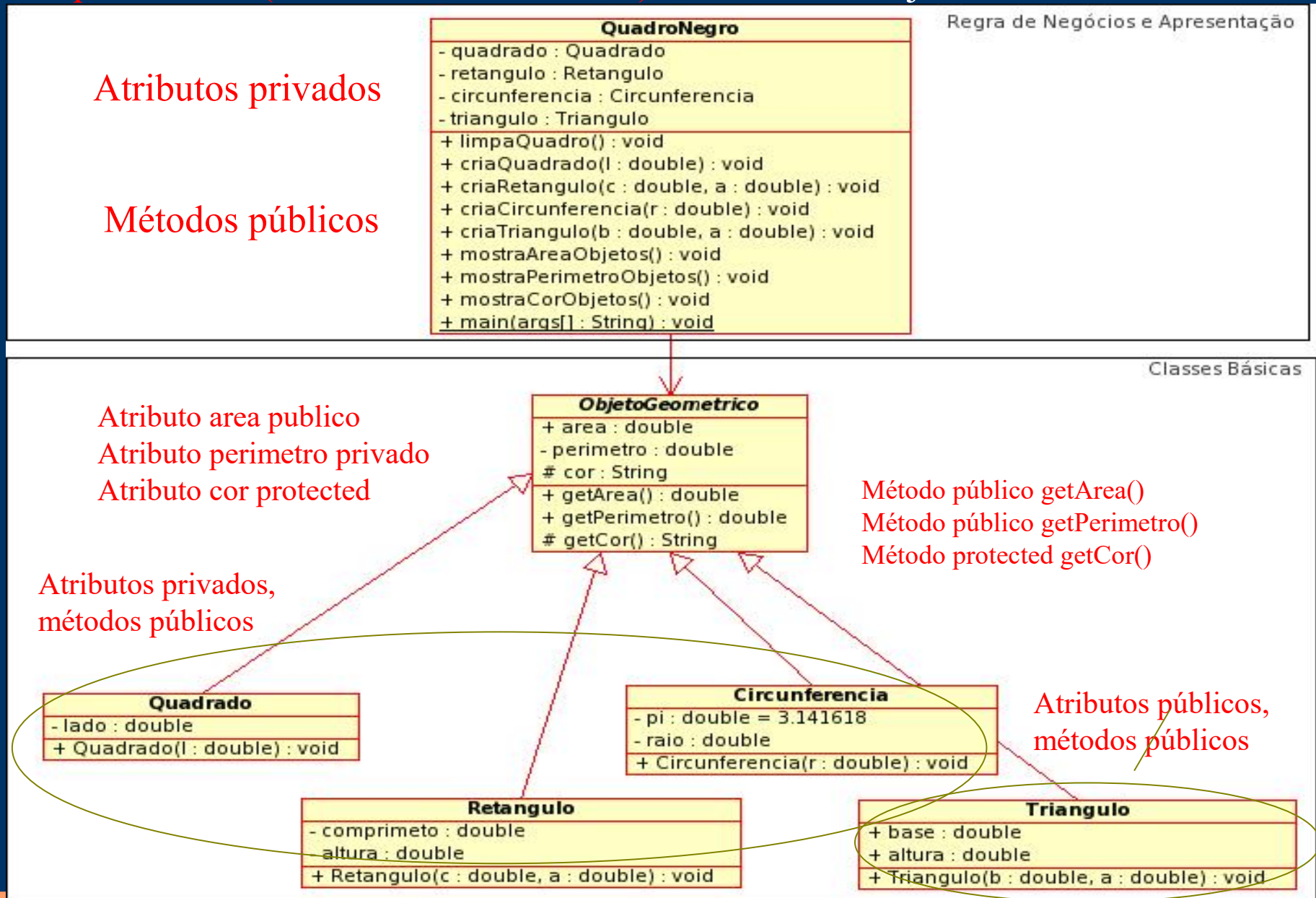
Definições Paradigma OO

- ✓ Encapsulamento (moderadores de acesso)
 - ✓ Exemplos de moderadores de acesso :
 - ✓ public – acessível por todos os métodos de qualquer outra classe
 - ✓ protected – acessíveis às classes pertencentes a um dado pacote (package)
 - ✓ friendly – acessível somente às classes derivadas da classe dada
 - ✓ private – acessível somente à classe em que o método foi definido
 - ✓ private protected – O método é acessível pela classe que o contém e por classes derivadas desde que definidas no mesmo código-fonte
-
-

Definições Paradigma OO

✓ Encapsulamento (moderadores de acesso)

Projeto Geometria



Definições Paradigma OO

✓ Encapsulamento (moderadores de acesso)

Projeto Geometria

```
public class QuadroNegro {  
    private Quadrado quadrado;  
    private Retangulo retangulo;  
    private Circunferencia circunferencia;  
    private Triangulo triangulo;  
  
    public void limpaQuadro ( ) {  
  
    }  
  
    public void criaQuadrado ( double l) {  
        quadrado = new Quadrado(l);  
    }  
  
    public void criaRetangulo ( double c, double a) {  
        retangulo = new Retangulo(c, a);  
    }  
  
    public void criaCircunferencia ( double r) {  
        circunferencia = new Circunferencia(r);  
    }  
  
    public void criaTriangulo ( double b, double a) {  
        triangulo = new Triangulo(b, a);  
    }  
}
```

Definições Paradigma OO

✓ Encapsulamento (moderadores de acesso)

Projeto Geometria

Os atributos
base e altura da
classe Triangulo
são públicos,
logo permitem o
acesso direto a
partir da classe
QuadroNegro

```
public void mostraAreaObjetos ( ) {  
    System.out.println("Quadrado. Área: " + quadrado.getArea());  
    System.out.println("Retângulo. Área: " + retangulo.getArea());  
    System.out.println("Triângulo. Área: " + (triangulo.base * triangulo.altura)/2);  
    System.out.println("Circunferência. Área: " + circunferencia.getArea());  
}  
  
public void mostraPerimetroObjetos ( ) {  
  
}  
  
public void mostraCorObjetos ( ) {  
  
}  
  
public static void main ( String args[] ) {  
    QuadroNegro quadroNegro = new QuadroNegro();  
    quadroNegro.criaQuadrado(10.6);  
    quadroNegro.criaRetangulo(50.5,20.4);  
    quadroNegro.criaTriangulo(6.7,5.5);  
    quadroNegro.criaCircunferencia(10);  
    quadroNegro.mostraAreaObjetos();  
  
}  
}
```


Definições Paradigma OO

✓ Encapsulamento (moderadores de acesso)

Projeto Geometria

```
abstract public class ObjetoGeometrico {  
    public double area;  
    private double perimetro;  
    protected String cor;  
  
}
```

Necessitará ser
alterado para
protected a fim de
que o atributo esteja
acessível a partir
das classes filha



Definições Paradigma OO

✓ Encapsulamento (moderadores de acesso)

Projeto Geometria

```
public class Quadrado extends ObjetoGeometrico {  
    private double lado;  
    Quadrado (double l) {  
        lado = l;  
  
    }  
  
    public double getArea ( ) {  
        area = lado * lado;  
        return area;  
    }  
  
}
```

Definições Paradigma OO

✓ Encapsulamento (moderadores de acesso)

Projeto Geometria

```
public class Retangulo extends ObjetoGeometrico {  
    private double comprimento;  
    private double altura;
```

```
    Retangulo(double c, double a) {  
        comprimento = c;  
        altura = a;  
  
    }
```

```
    public double getArea ( ) {  
        area = comprimento * altura;  
        return area;  
    }  
}
```

Definições Paradigma OO

✓ Encapsulamento (moderadores de acesso)

Projeto Geometria

```
public class Triangulo extends ObjetoGeometrico {  
    public double base;  
    public double altura;  
    Triangulo(double b, double a) {  
        base = b;  
        altura = a;  
  
    }  
    public double getArea ( ) {  
        area = base * altura /2;  
        return area;  
    }  
}
```


Definições Paradigma OO

✓ Encapsulamento (moderadores de acesso)

Projeto Geometria

```
public class Circunferencia extends ObjetoGeometrico {  
    private double pi = 3.141618;  
    private double raio;  
    Circunferencia(double r) {  
        raio = r;  
    }  
    public double getArea ( ) {  
        area = pi * (raio * raio);  
        return area;  
    }  
}
```

Definições Paradigma OO

- ✓ Encapsulamento (moderadores de acesso) Projeto Geometria
 - ✓ Exercício:
 - ✓ Implemente os métodos getPerimetro() e getCor() nas classes Quadrado, Retângulo, Triângulo e Circunferência.
 - ✓ Em seguida, implemente os métodos mostraPerimetroObjetos() e mostraCorObjetos() na classe QuadroNegro.
 - ✓ Por fim, no método main da classe QuadroNegro, mostre na tela as novas funcionalidades implementadas para os objetos
 - ✓ Dica: Será necessário definir as cores dos objetos para se poder acessar através de métodos de acesso as cores definidas para cada objeto geométrico criado. Para isso, crie o método setCor(String cor) na classe ObjetoGeometrico
-
-

Definições Paradigma OO

✓ Resposta do Exercício

Projeto Geometria

✓ Implementação dos métodos getPerimetro() e getCor() nas classes Quadrado, Retângulo, Triângulo e Circunferência.

✓ Quadrado

```
public class Quadrado extends ObjetoGeometrico {
```

Claro que além dos métodos já existentes, é preciso acrescentar este

```
    public double getPerimetro ( ) {  
        perimetro = 2 * lado;  
        return perimetro;  
    }
```

```
}
```

Definições Paradigma OO

- ✓ Resposta do Exercício
- ✓ Implementação dos métodos `getPerimetro()` e `getCor()` nas classes `Quadrado`, `Retângulo`, `Triângulo` e `Circunferência`.
- ✓ Retângulo

```
public class Retangulo extends ObjetoGeometrico {
```

Claro que além dos métodos já existentes, é preciso acrescentar este

```
    public double getPerimetro ( ) {  
        perimetro = comprimento + altura;  
        return perimetro;  
    }
```

```
}
```

Definições Paradigma OO

✓ Resposta do Exercício

Projeto Geometria

✓ Implementação dos métodos getPerimetro() e getCor() nas classes Quadrado, Retângulo, Triângulo e Circunferência.

✓ Triângulo

```
public class Triangulo extends ObjetoGeometrico {
```

Claro que além dos métodos já existentes, é preciso acrescentar este

```
    public double getPerimetro ( ) {  
        perimetro = 3 * base; // considerando equilátero  
        return perimetro;  
    }
```

```
}
```

Definições Paradigma OO

✓ Resposta do Exercício

Projeto Geometria

✓ Implementação dos métodos `getPerimetro()` e `getCor()` nas classes `Quadrado`, `Retângulo`, `Triângulo` e `Circunferência`.

✓ Circunferencia

```
public class Circunferencia extends ObjetoGeometrico {
```

Claro que além dos métodos já existentes, é preciso acrescentar este

```
    public double getPerimetro ( ) {  
        perimetro = 2 * pi * raio;  
        return perimetro;  
    }
```

```
}
```

Definições Paradigma OO

✓ Resposta do Exercício

Projeto Geometria

- ✓ Implementação dos métodos `getPerimetro()` e `getCor()` nas classes `Quadrado`, `Retângulo`, `Triângulo` e `Circunferência`.
- ✓ Observe que o método `getCor()` somente precisa ser implementado na classe `Pai` `ObjetoGeometrico`. As classes filhas herdam a definição.

É preciso alterar
perimetro para
protected ou public a
fim de estar acessível
nas classes filha

```
abstract public class ObjetoGeometrico {  
    public double area;  
    protected double perimetro;  
    protected String cor;
```

Claro que além do
já definido, é
preciso acrescentar
este método

```
    public String getCor ( ) {  
        return cor;  
    }
```

```
}
```

Definições Paradigma OO

✓ Resposta do Exercício

Projeto Geometria

✓ Implementação dos métodos mostraPerimetroObjetos() e mostraCorObjetos() na classe QuadroNegro.

```
public class QuadroNegro() {  
    public void mostraPerimetroObjetos() {  
        System.out.println("O perímetro dos objetos são: ");  
        System.out.println("Quadrado: " + quadrado.getPerimetro());  
        System.out.println("Retangulo: " + retangulo.getPerimetro());  
        System.out.println("Triângulo: " + triangulo.getPerimetro());  
        System.out.println("Circunferência: " + circunferencia.getPerimetro());  
    }  
}
```

Claro que além dos métodos já existentes, é preciso acrescentar este

Definições Paradigma OO

✓ Resposta do Exercício

Projeto Geometria

✓ Implementação dos métodos mostraPerimetroObjetos() e mostraCorObjetos() na classe QuadroNegro.

```
public class QuadroNegro() {  
    public void mostraCorObjetos() {  
        System.out.println("A cor dos objetos é: ");  
        System.out.println("Quadrado: " + quadrado.getCor());  
        System.out.println("Retangulo: " + retangulo.getCor());  
        System.out.println("Triângulo: " + triangulo.getCor());  
        System.out.println("Circunferência: " + circunferencia.getCor());  
    }  
}
```

Claro que além dos métodos já existentes, é preciso acrescentar este

Definições Paradigma OO

✓ Resposta do Exercício

Projeto Geometria

✓ Implementação dos métodos `mostraPerimetroObjetos()` e `mostraCorObjetos()` na classe `QuadroNegro`.

Alteração na chamada dos métodos `criaQuadrado`, `criaRetangulo`, `criaTriangulo` e `criaCircunferencia` da classe `QuadroNegro` para configurar também a cor dos objetos

Novos métodos da classe `QuadroNegro`: `mostraPerimetro()` e `mostraCor()`

```
public class QuadroNegro {  
  
    public static void main ( String args[] ) {  
  
        QuadroNegro quadroNegro = new QuadroNegro();  
        quadroNegro.criaQuadrado(10.6, "Branco");  
        quadroNegro.criaRetangulo(50.5,20.4, "Azul");  
        quadroNegro.criaTriangulo(6.7,5.5, "Amarelo");  
        quadroNegro.criaCircunferencia(10, "Verde");  
        quadroNegro.mostraPerimetro();  
        quadroNegro.mostraCor();  
  
    }  
}
```

Definições Paradigma OO

✓ Resposta do Exercício

Projeto Geometria

✓ Alteração dos métodos criaQuadrado, criaRetangulo, criaCircunferencia e criaTriangulo classe QuadroNegro.

```
public class QuadroNegro {
```

```
    public void criaQuadrado ( double l, String c) {  
        quadrado = new Quadrado(l);  
        quadrado.setCor(c);  
    }
```

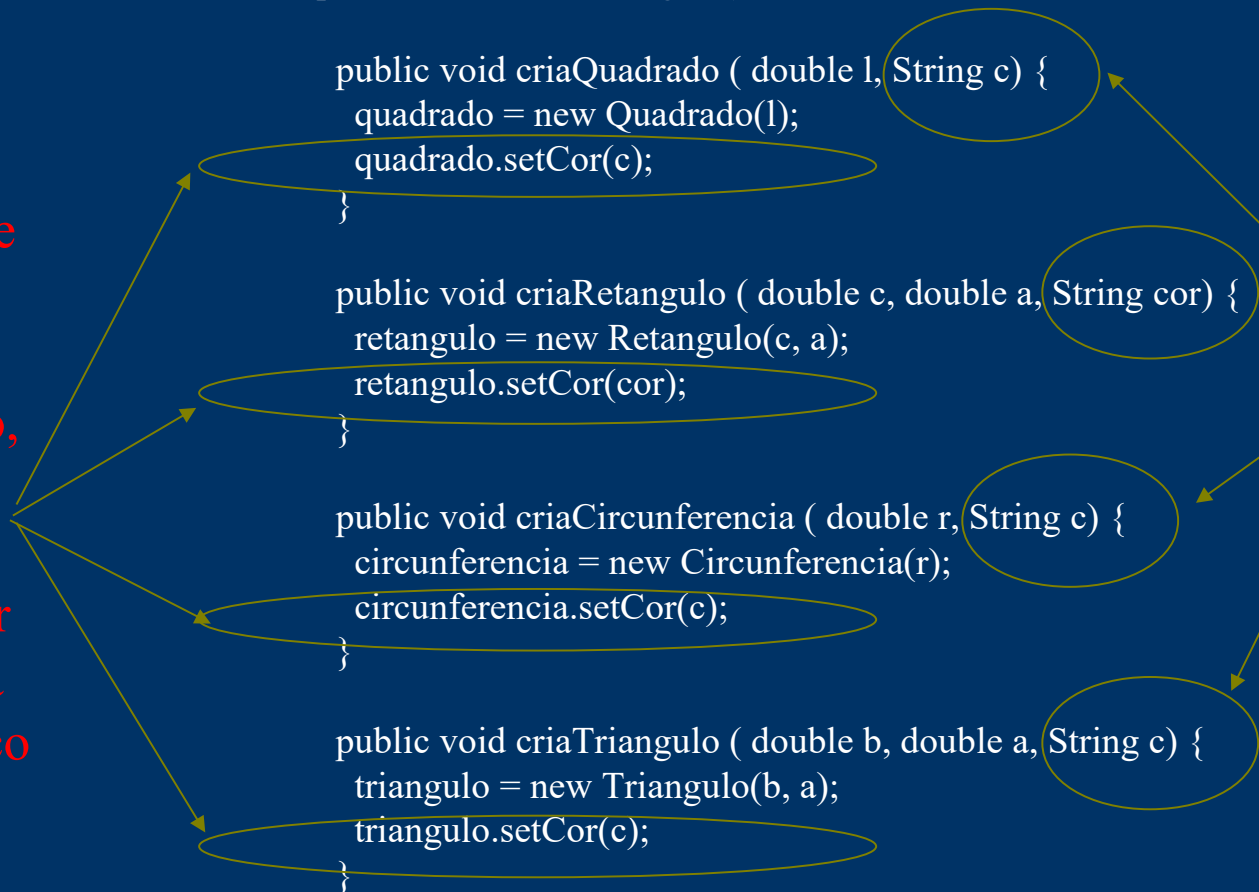
```
    public void criaRetangulo ( double c, double a, String cor) {  
        retangulo = new Retangulo(c, a);  
        retangulo.setCor(cor);  
    }
```

```
    public void criaCircunferencia ( double r, String c) {  
        circunferencia = new Circunferencia(r);  
        circunferencia.setCor(c);  
    }
```

```
    public void criaTriangulo ( double b, double a, String c) {  
        triangulo = new Triangulo(b, a);  
        triangulo.setCor(c);  
    }
```

Nova chamada de método setCor herdado pelas classes Quadrado, Retangulo, Circunferencia e Triangulo a partir da classe abstrata ObjetoGeometrico

Nova assinatura de método



Definições Paradigma OO

✓ Resposta do Exercício

Projeto Geometria

✓ Alteração dos métodos criaQuadrado, criaRetangulo, criaCircunferencia e criaTriangulo classe QuadroNegro.

Além do já
definido, é preciso
criar um novo
método em
ObjetoGeometrico:
setCor(String c)
Este método será
herdado pelas
classes Quadrado,
Retangulo,
Triangulo e
Circunferencia

```
abstract public class ObjetoGeometrico {
```

```
    public void setCor (String c) {  
        cor = c;  
    }
```

```
}
```

Modelo OO

Vantagens para Desenvolvimento de Software

- ✓ Pode ser usado desde o início do projeto, quando os conceitos são mais abstratos até a fase de implementação.
 - ✓ Dá mais importância à estrutura do sistema do que às funções.
 - ✓ Funções do sistema mudam
 - ✓ Estruturas raramente mudam (e.g. Universidade sempre terá professores)
-
-

Modelo OO

Vantagens para Desenvolvimento de Software

- ✓ Abstração
 - ✓ Noções de operação e interface permitem a concepção de classes abstratas, sem se preocupar com as suas implementações
 - ✓ Encapsulamento
 - ✓ Graças ao conceito de interface, o exterior da classe é protegido das modificações que podem ocorrer em seu interior (como é implementada)
 - ✓ Enquanto a interface não mudar a classe não muda para o exterior
-
-

Modelo OO

Vantagens para Desenvolvimento de Software

- ✓ Agrupamento de Dados e Funções
 - ✓ Uma classe possui dados e funções. Então, classes diferentes podem ter várias implementações de uma mesma operação. Cada implementação é diferente de uma para outra mas todas pertencem à mesma implementação. Isto é chamado de **Polimorfismo** que é uma das características fundamentais de OO
-
-

Modelo OO

Vantagens para Desenvolvimento de Software

- ✓ Reutilização de código através:
- ✓ Herança que permite definir uma nova classe reutilizando outras já definidas
- ✓ Aproveitamento das classes em outros sistemas. Uma classe é um pequeno módulo com dados e funções

Referências

1. Desenvolvimento de Softwares Orientados a Objetos. Prof. Nicolas Anquetil. Disponível em <http://www.ucb.br/ucbtic/mgcti/paginapessoalprof/Nicolas/Disciplinas/UML/uml.pdf>
2. UML Guia do Usuário. Grady Booch, James Rumbaugh, Ivar Jacobson.. Editora Campus, 2000.
3. JAVA Como Programar. H. M. Deitel, P. J. Deitel. Editora Bookman, 2001.
4. **The Unified Software Development Process. Ivar Jacobson, Grady Booch, James Rumbaugh. Addison Wesley, 1998.**
5. Página da Rational. Disponível em <http://www.rational.com>
6. Página do OMG. Disponível em <http://www.omg.org>
7. **JAVA 1001 Dicas de Programação. Mark C. Chan, Steven W. Griffith e Anthony F. Iasi. Makron Books 1999**