

Programação OO

V

II

Parte II



UNIVERSIDADE
VILA VELHA
ESPÍRITO SANTO

Métodos e Classes Abstratas

Hoje nós vamos ampliar o nosso conhecimento em OO, conversando sobre:

- Métodos e Classes Abstratas



Métodos e Classes Abstratas

É chamada de abstração a capacidade de:

- Determinar o problema de maneira geral, dando importância apenas aos seus aspectos relevantes e ignorando os detalhes

Ao criarmos uma classe para ser estendida,

- Às vezes codificamos alguns métodos para os quais não sabemos dar uma implementação, ou seja,
 - Um método que só subclasses saberão implementar

Uma classe deste tipo não pode ser instanciada pois sua funcionalidade está incompleta.

- É como se tentássemos andar de carro sem ter as rodas construídas....



Métodos e Classes Abstratas

Resumindo: Uma classe abstrata é utilizada quando deseja-se:

- Fornecer uma interface comum a diversos membros de uma hierarquia de classes.
- Não se preocupar COMO, apenas QUAL o contrato que deve ser obedecido...



Métodos e Classes Abstratas

Como isso fica?? Java suporta o conceito:

- De classes abstratas:
 - Podemos declarar uma classe abstrata usando o modificador *abstract*
- De métodos abstratos:
 - Usamos o modificador *abstract* e omitimos a implementação desse método.
- Vamos analisar um exemplo....



Java – Métodos e Classes Abstratas

Método sem
corpo { }

```
abstract class Veiculo  
{  
    abstract void Abastece ();  
}
```

```
class Onibus extends Veiculo  
{  
    void Abastece ()  
    {  
        EncheTanqueComOleoDiesel();  
    }  
}
```

...

```
Veiculo v = new Veiculo(); // Erro!
```

Características

- Abstração : suprimir detalhes para melhor compreensão
- Classe abstrata - possui métodos abstratos
- Força a implementação para que possa ser instanciada

Todos os veículos possuem a ação de abastecer, mas a forma de abastecer (álcool, gasolina ou diesel) só é conhecida pelas filhas da herança



UNIVERSIDADE
VILA VELHA
ESPÍRITO SANTO

Java – Métodos e Classes Abstratas

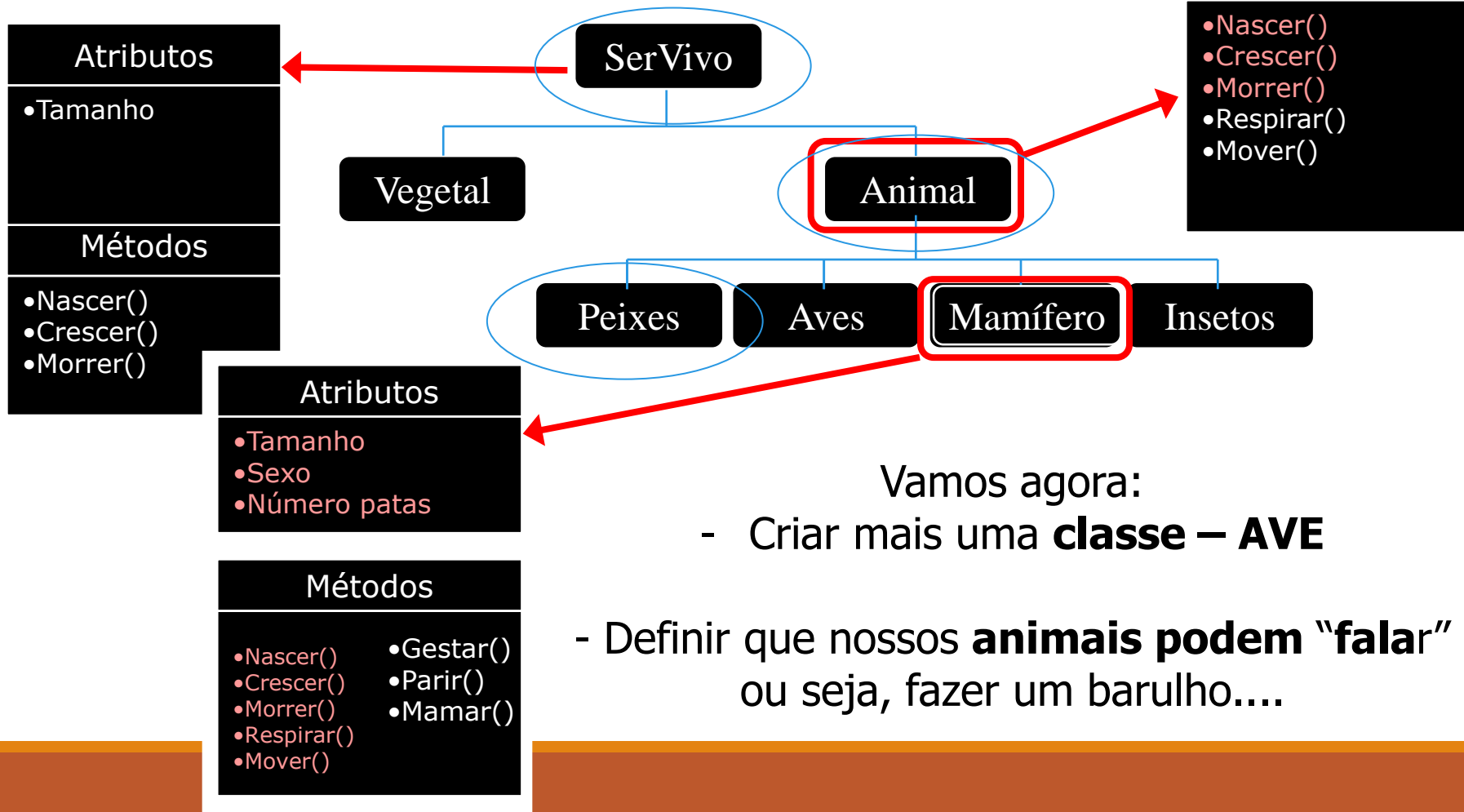
Blz

- Vamos considerar o exemplo a seguir e trabalhar com o polimorfismo de objetos....



Exercício – Vamos implementar juntos

Cenário modelado (*com "liberdade poética".. 😊*)





UNIVERSIDADE
VILA VELHA
ESPÍRITO SANTO

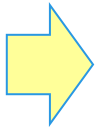
1/2

Exerício – Implementem agora...

O programa BichosIncompleto ilustra a definição de uma classe abstrata Bicho.

Para derivar esta classe, é necessário fornecer o som emitido por cada tipo de bicho (método som()).

Complete o programa criando as classes derivadas Gato e Cachorro, que devem respectivamente emitir os sons "Miau" e "Auau".





Exercício

```
abstract class Bicho {  
    protected String nome;  
  
    public Bicho() {  
    }  
  
    Bicho(String nome) {  
        this.nome = nome;  
    }  
  
    abstract protected String som();  
  
    public void fala() {  
        System.out.println(nome + ": " + som() + "!");  
    }  
}  
  
public class BichosIncompleto {  
    public static void main(String[] args) {  
        Gato g = new Gato("Garfield");  
        Cachorro c = new Cachorro("Pluto");  
        g.fala();  
        c.fala();  
    }  
}
```



UNIVERSIDADE
VILA VELHA
ESPÍRITO SANTO

Interfaces

Blz, pra fechar vamos conversar agora sobre....

- Herança Múltipla e Interfaces



Herança Múltipla

Em algumas linguagens de programação OO as classes podem ter mais de um superclasse....

- Herdando variáveis e métodos combinados de todas essas superclasses. Isto é chamado de herança múltipla

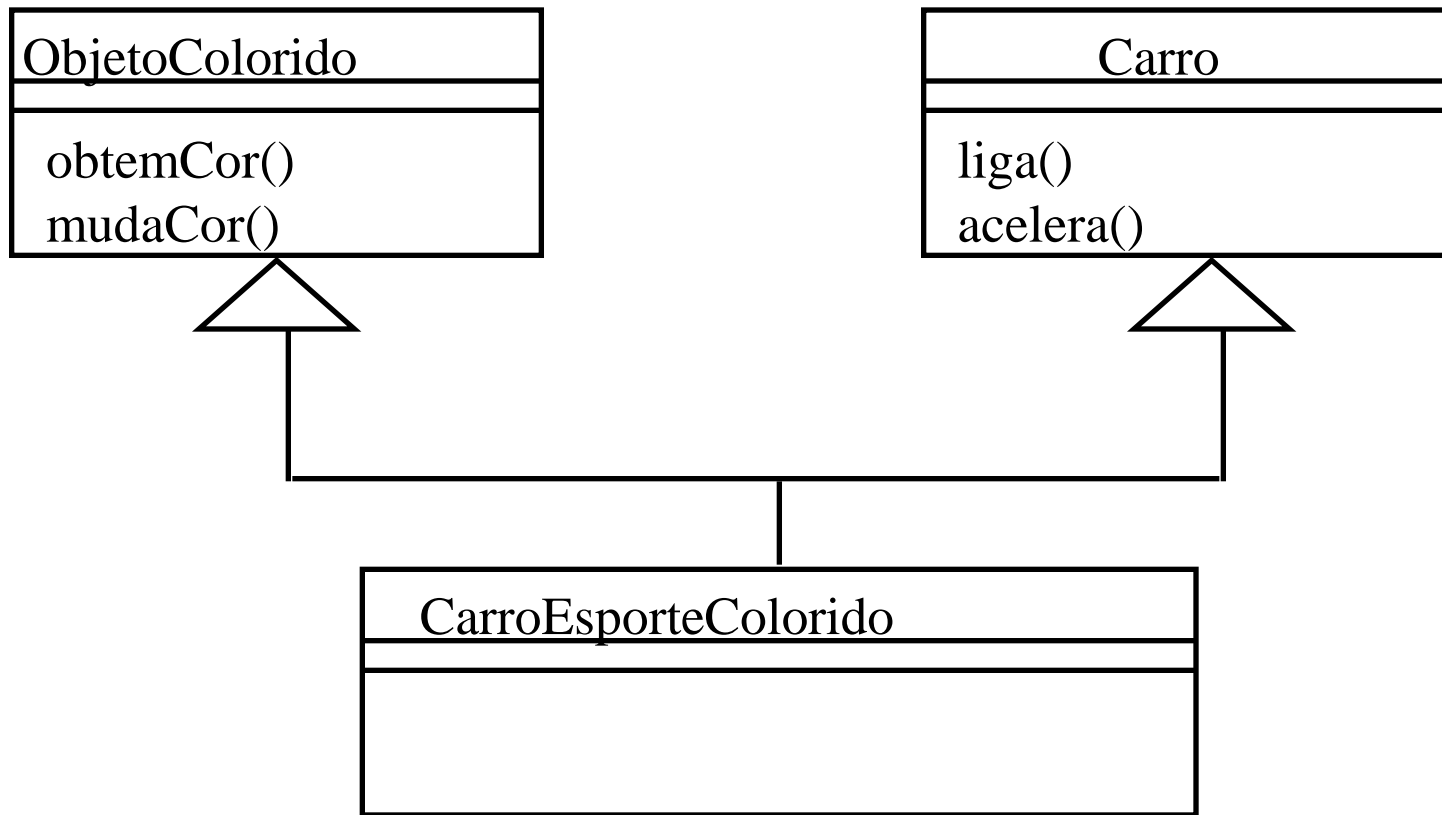
Herança Múltipla fornece margem para praticamente qualquer comportamento imaginável

- Contudo complica bastante as definições de classes e o código para produzi-la, além do entendimento



Herança Múltipla

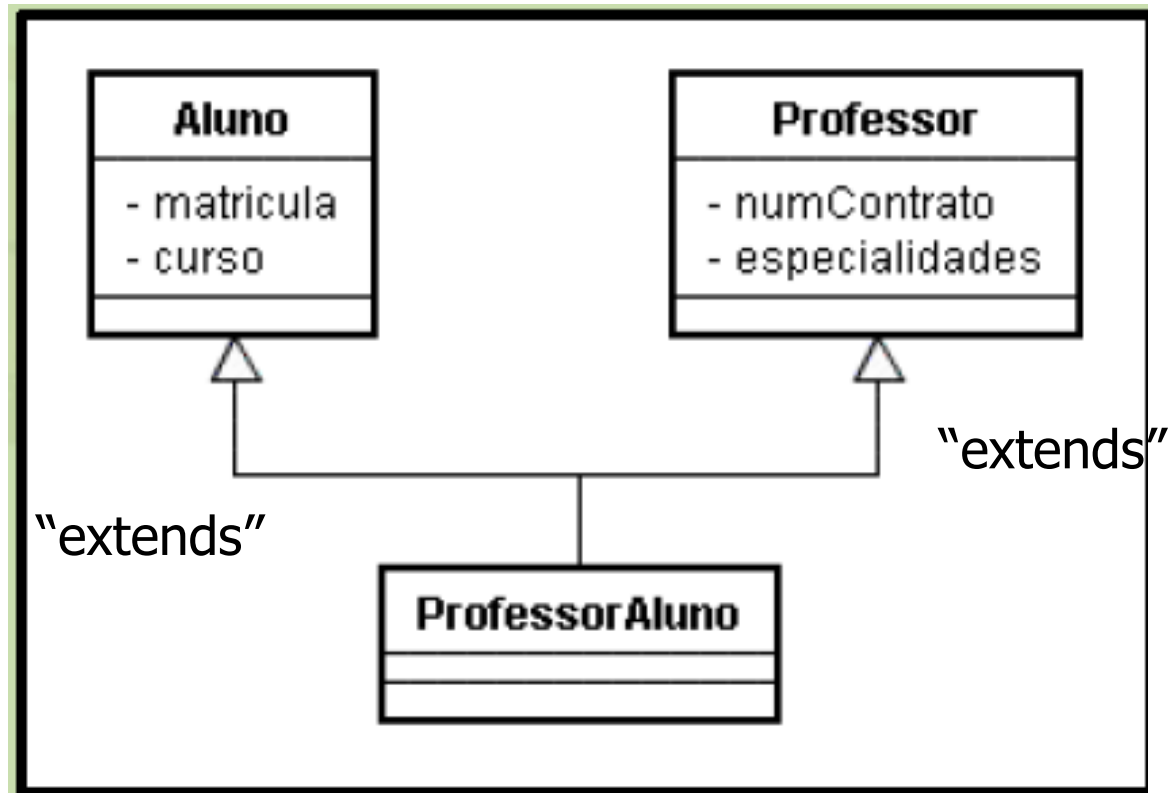
Exemplo de utilização de herança múltipla





Herança Múltipla

Outro exemplo de utilização herança múltipla





Herança Múltipla

Java simplifica a herança

- Permitindo diretamente apenas a herança simples ou única das classes

NÃO é permitido múltipla herança de classe,

- Pois foi verificado em outras linguagens que sua implementação pode ser muito confusa

Mas se tivermos que implementar essa modelagem, como podemos construir essa característica?



Interfaces

Os objetos podem implementar uma interface....

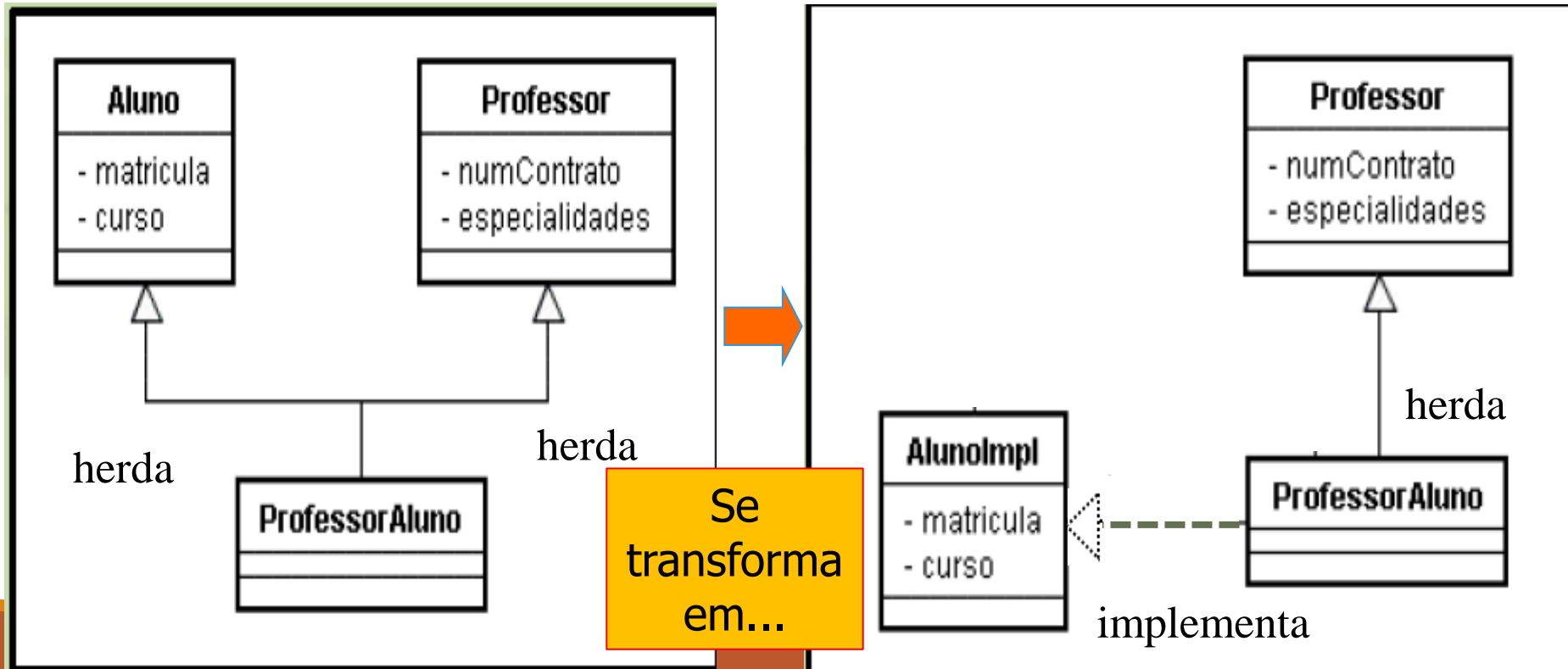
- As interfaces são como classes, mas não têm nenhuma implementação.
 - Define apenas as assinaturas de operações externamente visíveis que uma classe pode implementar, sem conter nenhuma especificação ou estrutura interna
- Na prática representa:
 - Um comportamento compartilhado por várias classes
 - Onde cada classe deve implementar os métodos



Java – Herança Múltipla

Solução para herança múltipla

- Escolha uma das classes para herdar;
- Para as demais, crie interfaces e implemente-as;
- Use composição ou agregação para fazer reuso.






Java - Class x Interface


Sintaxe para Classe e Interface em Java

```
public class MinhaClasse  
{  
    <construtor, métodos, atributos>  
}
```

```
public interface MinhaInterface  
{  
    <métodos sem implementação, atributos constantes>  
}
```



Interfaces não permitem declaração de atributos variantes (somente estáticos), enquanto que classes permitem outros.



Interfaces estão mais ligadas a comportamento, enquanto que classes estão mais ligadas a implementação.



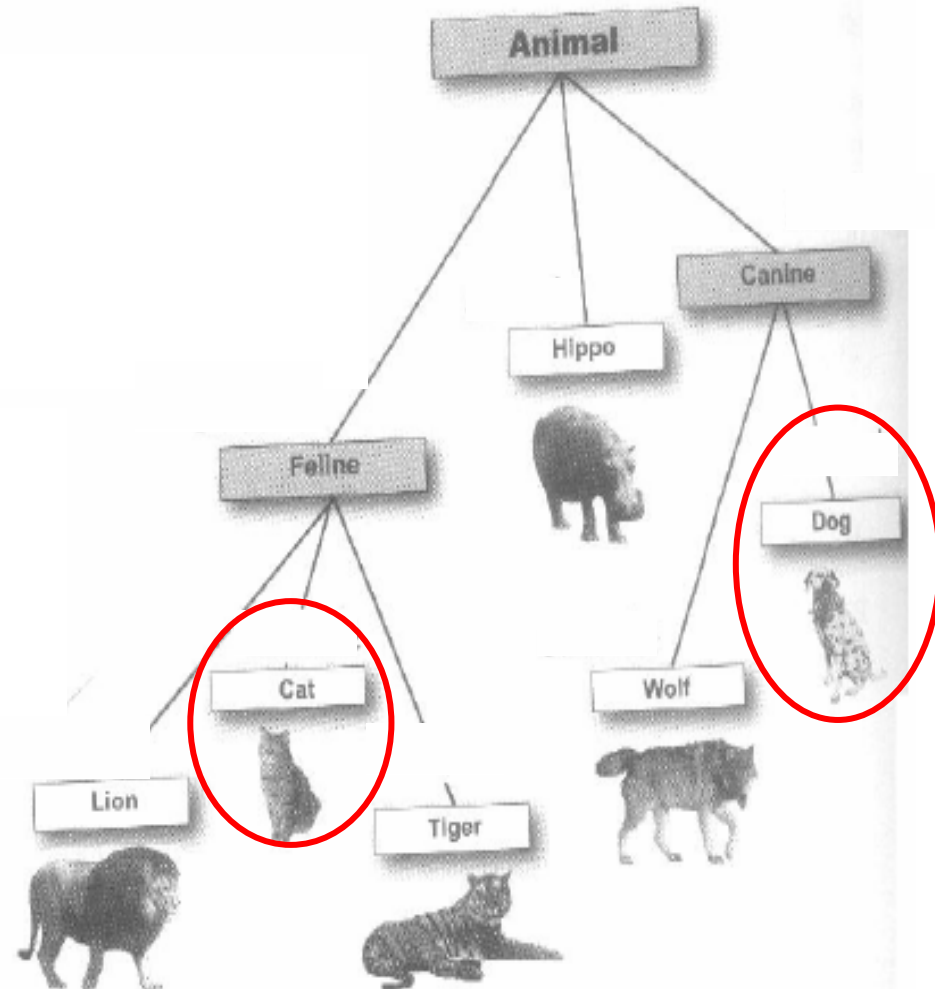
Interface ou classe abstrata?

Imagine que temos um Hierarquia de Animais, e devemos simular animais em um ambiente.

Se analisarmos as relações ao lado, a classe Dog ou Cat para um programa de ciências está OK....

.. Mas e para um programa que simule um Pet Shop???

- Ou seja precisamos de métodos específicos para isso...

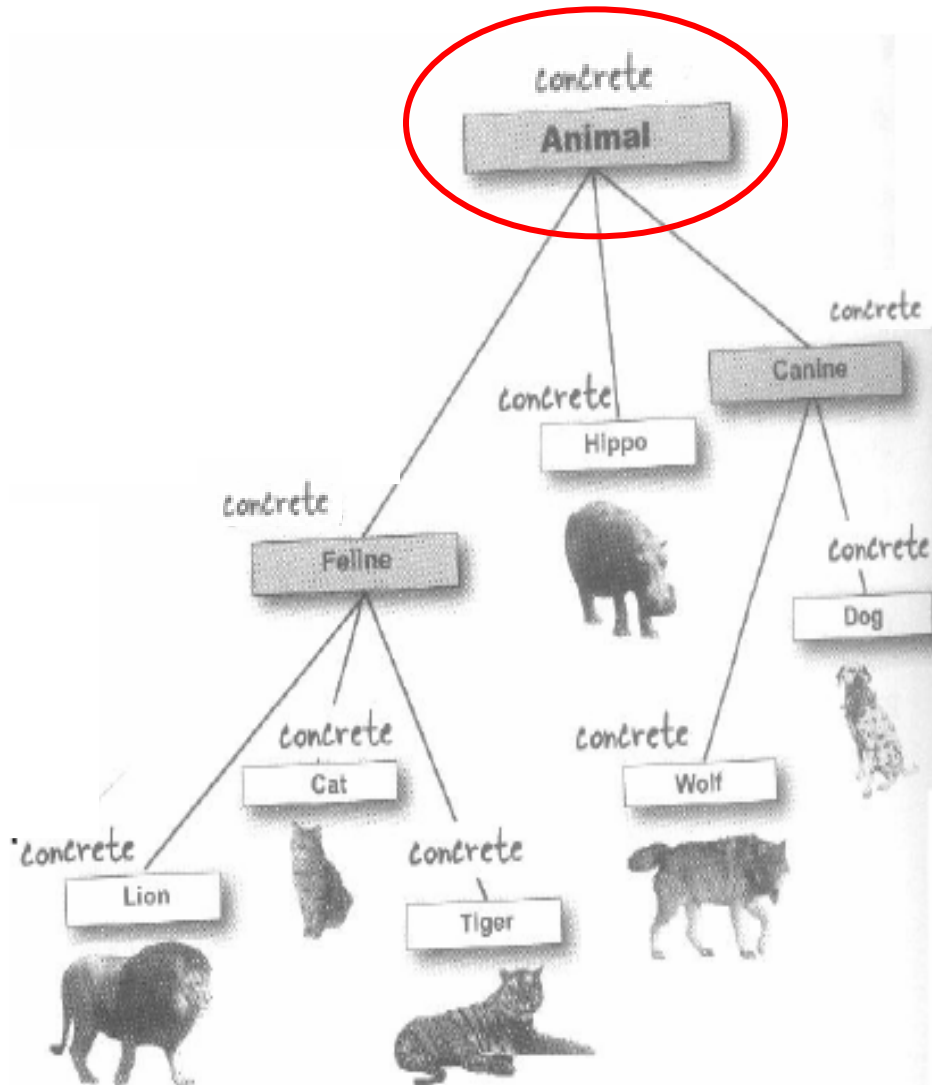




Interface ou classe abstrata?

1ª Opção: Métodos concretos que simulam os comportamentos de um Pet (Animal de Estimação) implementados em Animal.

- Prós:
 - Todas as subclasses possuirão esses comportamentos.
 - Uso do polimorfismo.
- Contras:
 - Existem subclasses de Animal que não precisam desses comportamentos (Hippo, Lion, Tiger, Wolf)

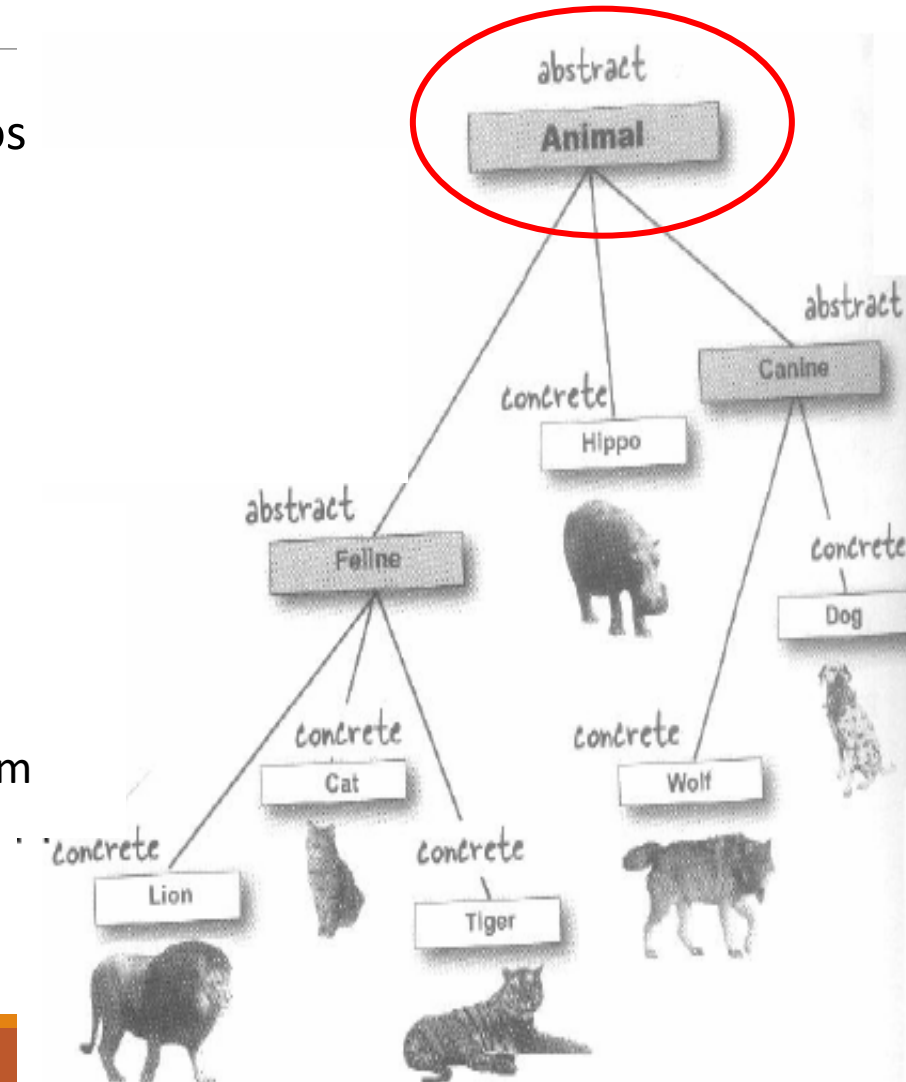




Interface ou classe abstrata?

2ª Opção: Métodos abstratos que simulam os comportamentos de um Pet (Animal de Estimação) definidos em Animal.

- Prós:
 - Todas as subclasses possuirão esses comportamentos;
 - Uso do polimorfismo;
 - Cada subclasse com seu próprio comportamento.
- Contras:
 - Obriga a todas as subclasses concretas devem implementar os métodos abstratos.

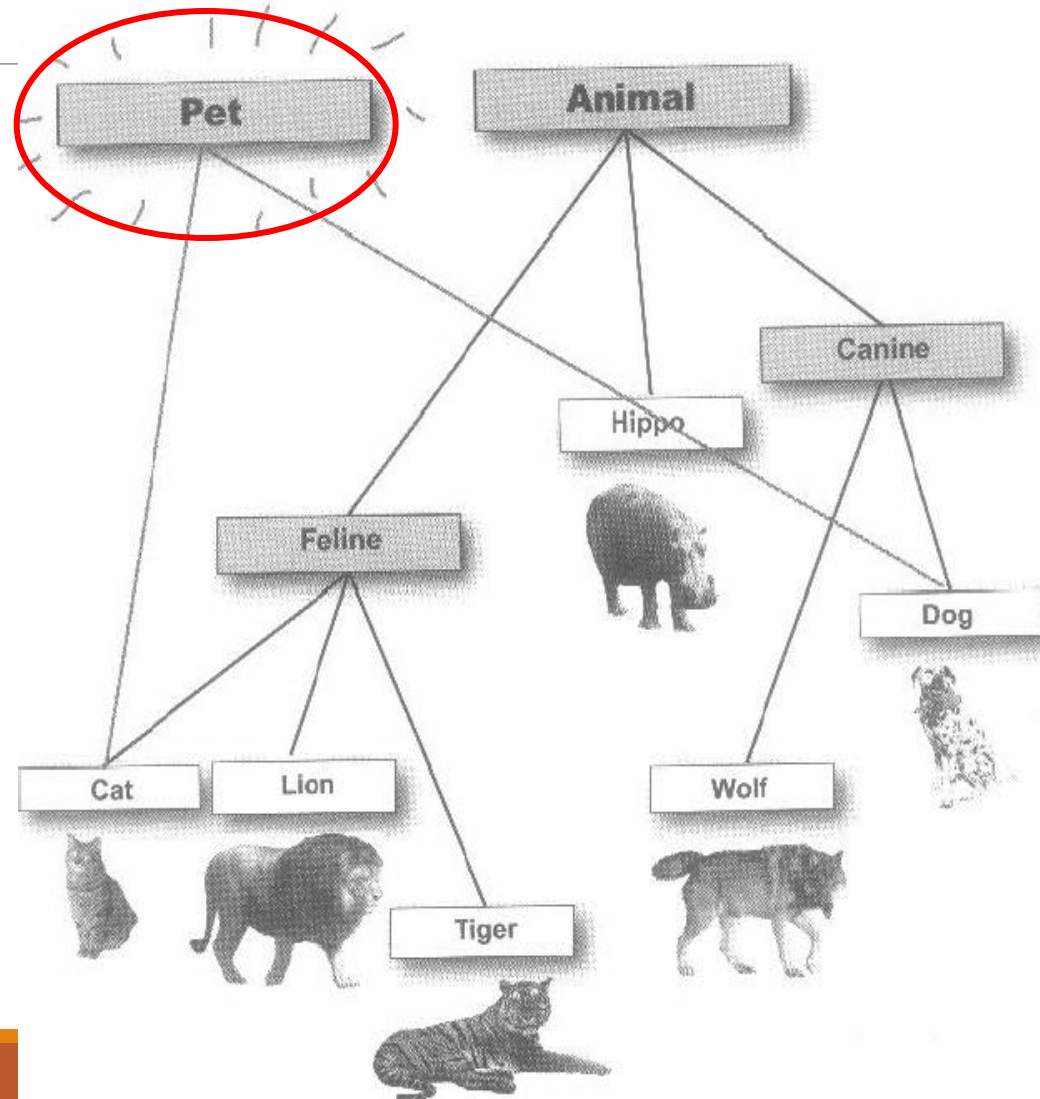




Interface ou classe abstrata?

3ª Opção: Nova classe com comportamentos de animais de estimação

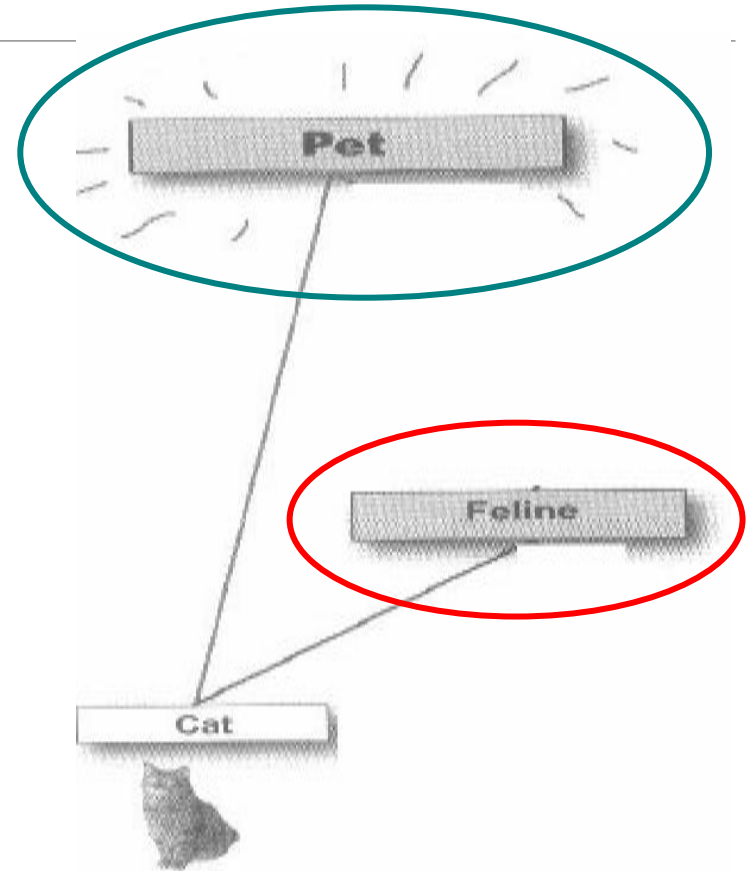
- Prós:
 - Garante que todos os animais de estimação terão as mesmas definições de métodos;
- Polimorfismo
- Contras:
 - Duas Superclasses;
 - Não suportado pela linguagem





Interface ou classe abstrata?

4ª Opção: Uma interface com comportamentos de animais de estimação





Interface ou classe abstrata?

Solução: Interface

- É como uma classe abstrata, mas só tem:
 - Métodos abstratos, públicos
 - Campos finais estáticos, públicos

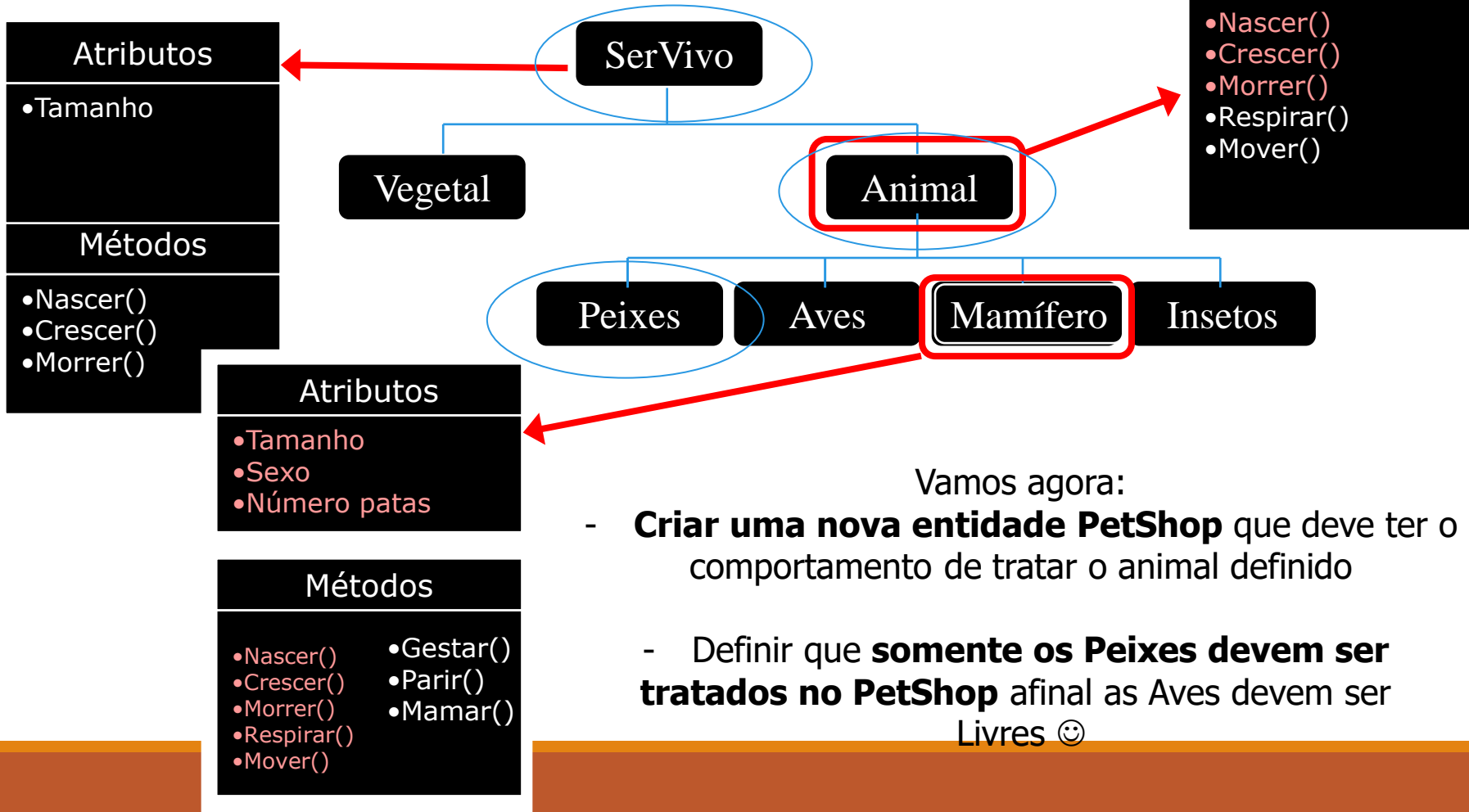
```
public interface Pet {  
    public abstract void beFriendly();  
    public abstract void play();  
}
```

```
public class Dog extends Canine implements Pet {  
    public void beFriendly(){ ... }  
    public void play(){ ... }  
  
    public void roam(){ ... }  
    public void eat(){ ... }  
}
```



Exercício

Cenário modelado (com "liberdade poética"..☺)



Vamos agora:

- **Criar uma nova entidade PetShop** que deve ter o comportamento de tratar o animal definido
- Definir que **somente os Peixes devem ser tratados no PetShop** afinal as Aves devem ser Livres ☺



Interfaces e Heranças

Outro conceito importante é que Interfaces suportam herança, mesmo herança múltipla.

- Interfaces podem herdar de outras interfaces, e suportam herança múltipla (enquanto classes suportam herança simples);
- Mecanismo para prover flexibilidade devido à ausência de herança múltipla de classes;



Java - Interface

Interfaces suportam herança, mesmo herança múltipla.

Interfaces podem herdar de outras interfaces, e suportam herança múltipla (enquanto classes suportam herança simples);

Hierarquias de interfaces

- Uma interface pode estender outra
- Uma classe pode implementar várias interfaces

```
public interface Ciclista extends Pessoa  
{  
    public abstract void Pedala (int km);  
}
```

Pessoa é uma outra interface
Método *Pedala* é *abstract*

```
Class Triatleta extends Atleta implements Ciclista, Corredor, Nadador  
{ ...  
}
```

O método *Pedala* deve ser implementado nesta classe



Exercício

1. Crie um projeto e copie o conteúdo da classe do próximo slide para um arquivo chamado ***Mostra.java***;
2. Compile e execute o código;
3. Analise os resultados;
4. Retire os marcadores de comentários “//”, e compile;
5. Por que o código não compila?
6. Utilizando ***implements***, o que pode ser feito para que o método ***Nome*** das classes Cicrano e Beltrano sejam evocadas?
7. Utilizando ***herança simples***, o que pode ser feito para que o método ***Nome*** das classes Cicrano e Beltrano sejam evocadas?
8. Sempre podemos optar por ***herança simples*** ou ***implements***?

```
class Fulano
{
    public String Nome()
    {
        return "Fulano da Silva";
    }
}
```

```
class Cicrano
{
    public String Nome()
    {
        return "Cicrano da Veiga";
    }
}
```

```
class Beltrano
{
    public String Nome()
    {
        return "Beltrano da Slovinsky";
    }
}
```

```
class Mostra
{
    static void MostraNome(Fulano r)
    {
        System.out.println("***> "+r.Nome()+" <***");
    }

    public static void main(String[] arg)
    {
        Fulano f=new Fulano();
        Cicrano c=new Cicrano();
        Beltrano b=new Beltrano();

        MostraNome(f);
        //MostraNome(c);
        //MostraNome(b); } }
```



UNIVERSIDADE
VILA VELHA
ESPÍRITO SANTO

Exercícios

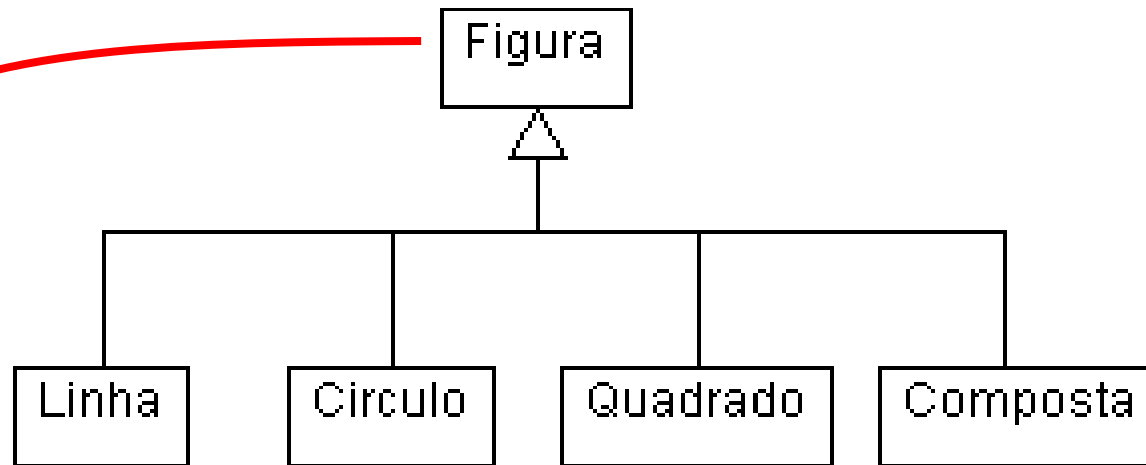
Blz... Agora é hora de exercitar.....

Tente resolver ou analisar os seguintes problemas...

- Em dupla
- Apresentar ao professor no final da aula
- Pontuação em Atividades em sala de aula...
- Faça o JAVADOC de todos os exercícios!!!



Exercício Junto....



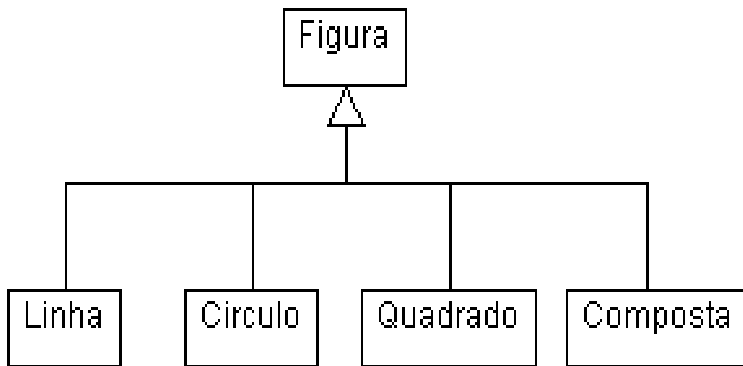
```
protected abstract void desenha(String cor);

protected void apaga(String mesmaCor){
    System.out.println("Apagando Desenho:" + mesmaCor);
    desenha(mesmaCor);
}
```





Exercício Junto....



```

class Composta extends Figura{
    Figura f1, f2;

    public Composta(Figura f1, Figura f2){
        this.f1 = f1;
        this.f2 = f2;
    }

    protected void desenha(String cor){
        System.out.println("Composta Desenha: " + cor);
        System.out.print("\t"); f1.desenha(cor);
        System.out.print("\t"); f2.desenha(cor);
    }

    public void composta3D(){
        System.out.println("Composta Desenha 3D.");
    }
}
  
```

```

class Circulo extends Figura{
    protected void desenha(String cor){
        System.out.println("Circulo Desenha: " + cor);
    }
}
  
```





Exercício Junto....

```
public static void main(String[] args){  
    Figura l = new Linha();  
    Figura c = new Circulo();  
    Figura q = new Quadrado();  
    Figura cp = new Composta(c,q);  
  
    System.out.println("=====");  
    l.desenha("Preto");  
  
    System.out.println("=====");  
    c.desenha("Azul");  
  
    System.out.println("=====");  
    q.desenha("Vermelho");  
  
    System.out.println("=====");  
    cp.desenha("Verde");  
  
    System.out.println("=====");  
    if (cp instanceof Composta)  
        ((Composta)cp).composta3D();  
}
```



Exercício

Implemente e verifique qual a saída após a execução do código abaixo...

```
interface Instrumento {
    int i = 5;
    void tocar();
    String nome();
    void afinar();
}

class Sopro implements Instrumento {
    public void tocar() {
        System.out.println("Sopro.tocar()");
    }
    public String nome() { return "Sopro"; }
    public void afinar() {}
}

class Percussao implements Instrumento {
    public void tocar() {
        System.out.println("Percussao.tocar()");
    }
    public String nome() { return "Percussao"; }
    public void afinar() {}
}
```

```
class Corda implements Instrumento {
    public void tocar() {
        System.out.println("Corda.tocar()");
    }
    public String nome() { return "Corda"; }
    public void afinar() {}
}

class SoproMetal extends Sopro {
    public void tocar() {
        System.out.println("SoproMetal.tocar()");
    }
    public void afinar() {
        System.out.println("SoproMetal.afinar()");
    }
}

class SoproMadeira extends Sopro {
    public void tocar() {
        System.out.println("SoproMadeira.tocar()");
    }
    public String nome() { return "SoproMadeira"; }
}
```





Exercício

Implemente e verifique qual a saída após a execução do código abaixo...

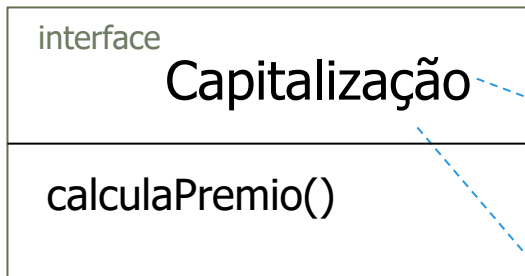
```
public class Musica {  
  
    static void melodia(Instrumento i) {  
        i.tocar();  
    }  
  
    static void sinfonia(Instrumento[] e) {  
        for(int i = 0; i < e.length; i++)  
            melodia(e[i]);  
    }  
  
    public static void main(String[] args) {  
        Instrumento[] orchestra = new Instrumento[5];  
        int i = 0;  
        // Upcasting - Polimorfismo  
        orchestra[i++] = new Sopro();  
        orchestra[i++] = new Percussao();  
        orchestra[i++] = new Corda();  
        orchestra[i++] = new SoproMetal();  
        orchestra[i++] = new SoproMadeira();  
        sinfonia(orchestra);  
    }  
}
```



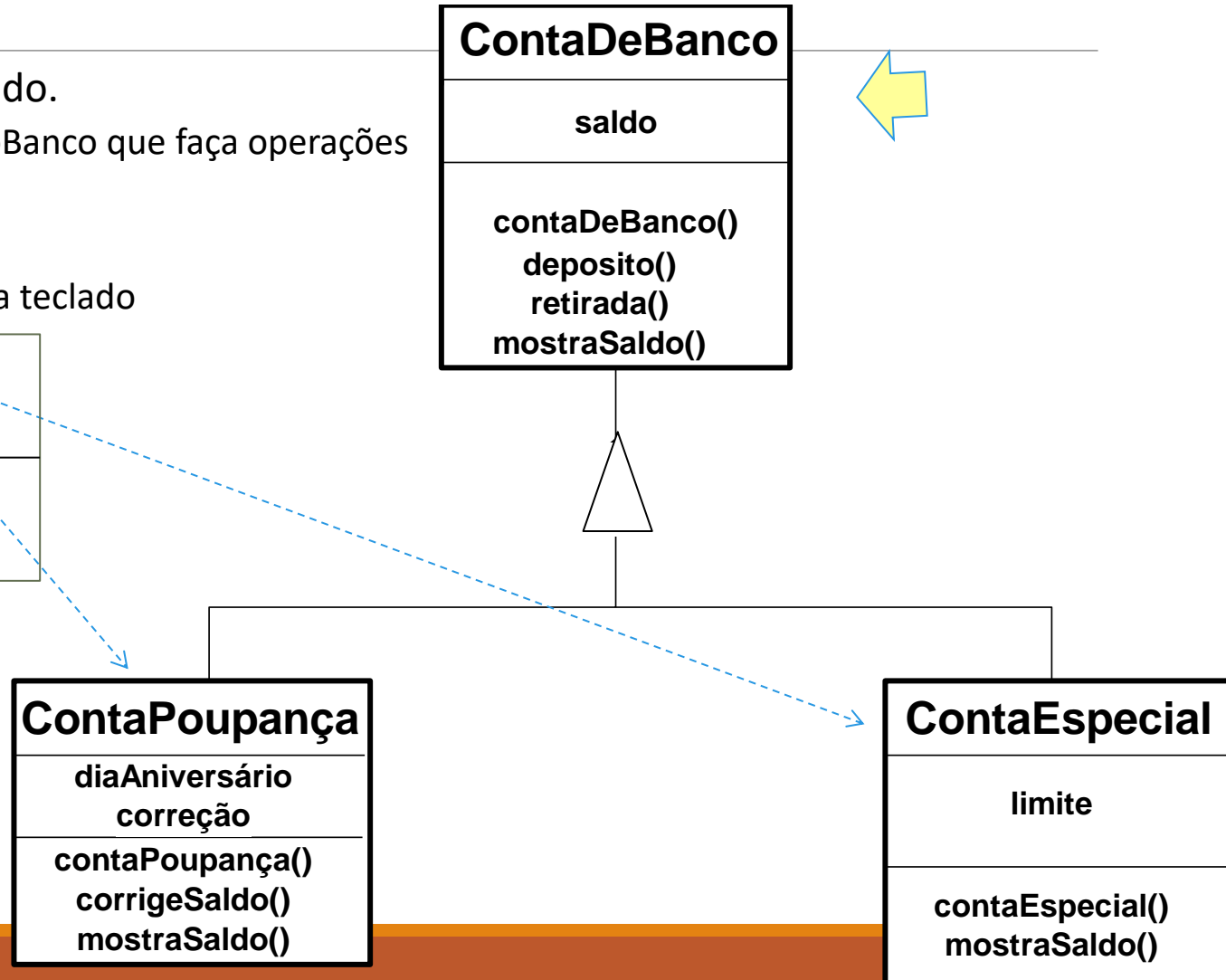
Exercício

Desenvolva as classes ao lado.

- Crie uma classe ExemploBanco que faça operações nesse banco.
- Pegue as informações via teclado



Pela correção ou pelo limite





Exercício

Criar uma estrutura hierárquica que contenha as seguintes classes:

- Veiculo (classe abstrata), Bicicleta e Automóvel.
- Os métodos da classe Veiculo são todos abstratos e possuem a seguinte assinatura:
 - listarVerificacoes()
 - ajustar()
 - limpar()
- Estes métodos são implementados nas subclasses Automóvel e Bicicleta.
- Acrescentar na classe Automóvel o método mudarOleo()





Exercício

Para desenvolver a classe Teste é necessário criar também a classe Oficina que terá dois métodos:

- proximo()
 - que retorna aleatoriamente um objeto do tipo bicicleta ou automóvel
- manutencao(Veiculo v)
 - que recebe como parâmetro um objeto do tipo veiculo e chama os métodos definidos na classe veiculo:
 - listarVerificacoes()
 - ajustar()
 - limpar()
 - se o veiculo for Automóvel deve também chamar o método mudarOleo()