

Programação OO

V

I

Parte I



Reuso / reutilização

Para entregar software de qualidade em menos tempo, é preciso reutilizar;

- Reuso é uma das principais vantagens anunciadas pela Orientação a Objetos;
- “Copy & paste” não é reuso!

É necessário o bom entendimento dos mecanismos de dependências de classes baseado no conceito de classes que são:

- Associação, Composição e Agregação
- Herança ou derivação



Dependências de classes

Até agora aprendemos a construir relacionamentos de: Composição, Agregação e Associação (“tem um”)

- Ex.: Uma conta tem um dono (cliente), etc..

As suas principais características são que:

- Usa objetos de outras classes
 - Uso de outra classe como atributo de uma nova classe
- Não altera comportamentos
 - Reuso como Cliente
- Cria dependências entre classes

Agora precisamos conversar sobre um outro tipo de relacionamento...



Herança

A outra forma de relacionamento entre as classes que temos em OO é a Herança,

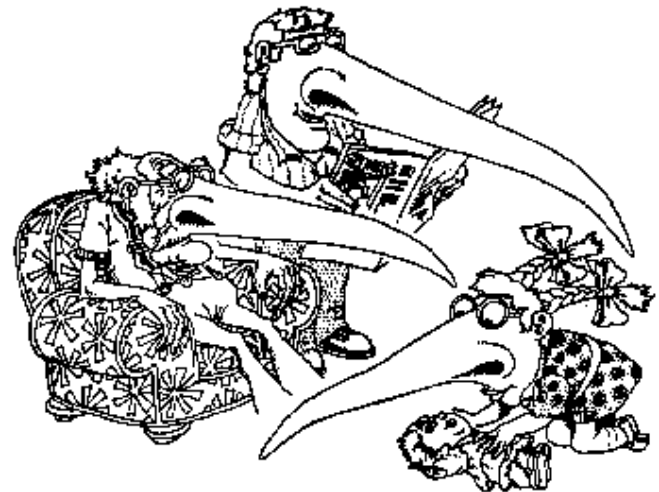
- É identificado como (“é-um”) ou (“é um tipo de”)

Define uma hierarquia de abstrações

- Na qual uma subclasse herda propriedades e comportamentos de uma superclasse

Vamos analisar o processo de herança dos seres humanos.

Lembrar do nariz do seu avô... 😊





Herança

E o que é herdado em uma classe em OO ?

- Atributos,
- Métodos
- e Relacionamentos

A classe derivada pode ainda:

- Adicionar atributos, métodos e relacionamentos
- Redefinir métodos
 - Vamos fazer um plástica no nariz... 😊



UNIVERSIDADE
VILA VELHA
ESPÍRITO SANTO

Generalização X Especialização

Generalização

"coisas" em
comum

Reformar
Limpar
Pintar
Mobiliar



CASA

Portas
Salas
Cozinha

Quartos
Localização
Telhado

(Superclasse)

PRAIA

(Subclasse)

FAVELA

MANSÃO

Limpar Piscina
Contratar Criadagem
Piscina
Quadras

"coisas" de
cada um

Especialização





Herança

Custos da Herança

- Velocidade da execução
 - Necessário identificar qual método no nível hierárquico que está se referindo
- Overhead de Mensagem
 - Há uma maior troca de mensagens devido aos níveis hierárquicos
- Isto não significa que não deve-se usar a herança,
 - Mas que deve-se compreender melhor os benefícios, e pesá-los em relação aos custos.



UNIVERSIDADE
VILA VELHA
ESPÍRITO SANTO

Herança

Blz,

- Isso tudo é bonito, mas como é que eu posso resolver os meus problemas em forma de herança??
- Como isso pode me ajudar??? Vamos analisar um exemplo de modelagem para entender como podemos pensar sobre a herança....



Herança

Seja o seguinte Contexto:

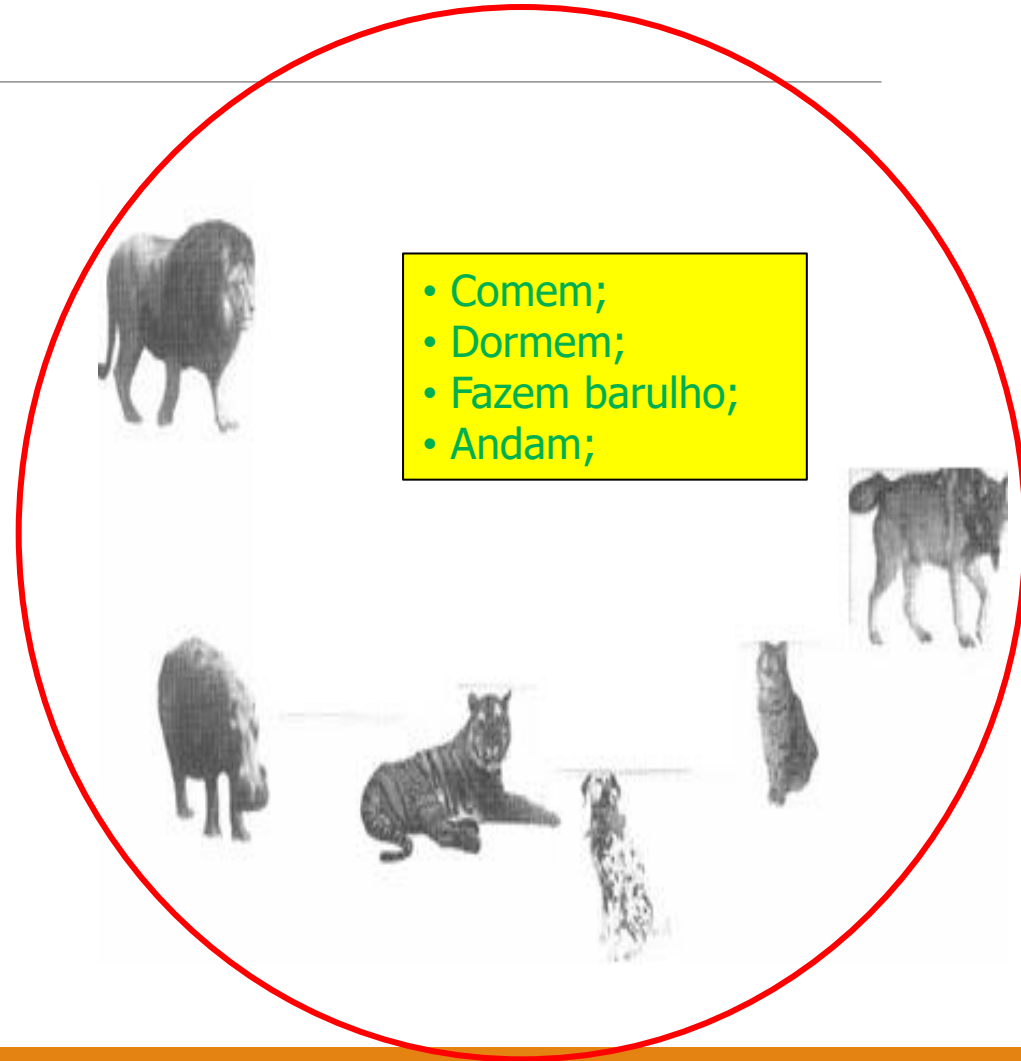
- Criar um programa para simular comportamento de vários animais em um ambiente;
- Possui conjunto de animais (não todos);
- Cada animal => 1 objeto;
- Cada animal move-se no ambiente a seu modo; fazendo qualquer coisa;
 - Ou seja, tem seu próprio comportamento também..
- Novos animais podem ser adicionados ao programa.



Herança

Passos para definir o relacionamento de Herança:

- 1º Passo:
 - Definir o que cada objeto animal tem em comum, como Atributos e Comportamentos;
 - Definir como esses tipos de animais se relacionam

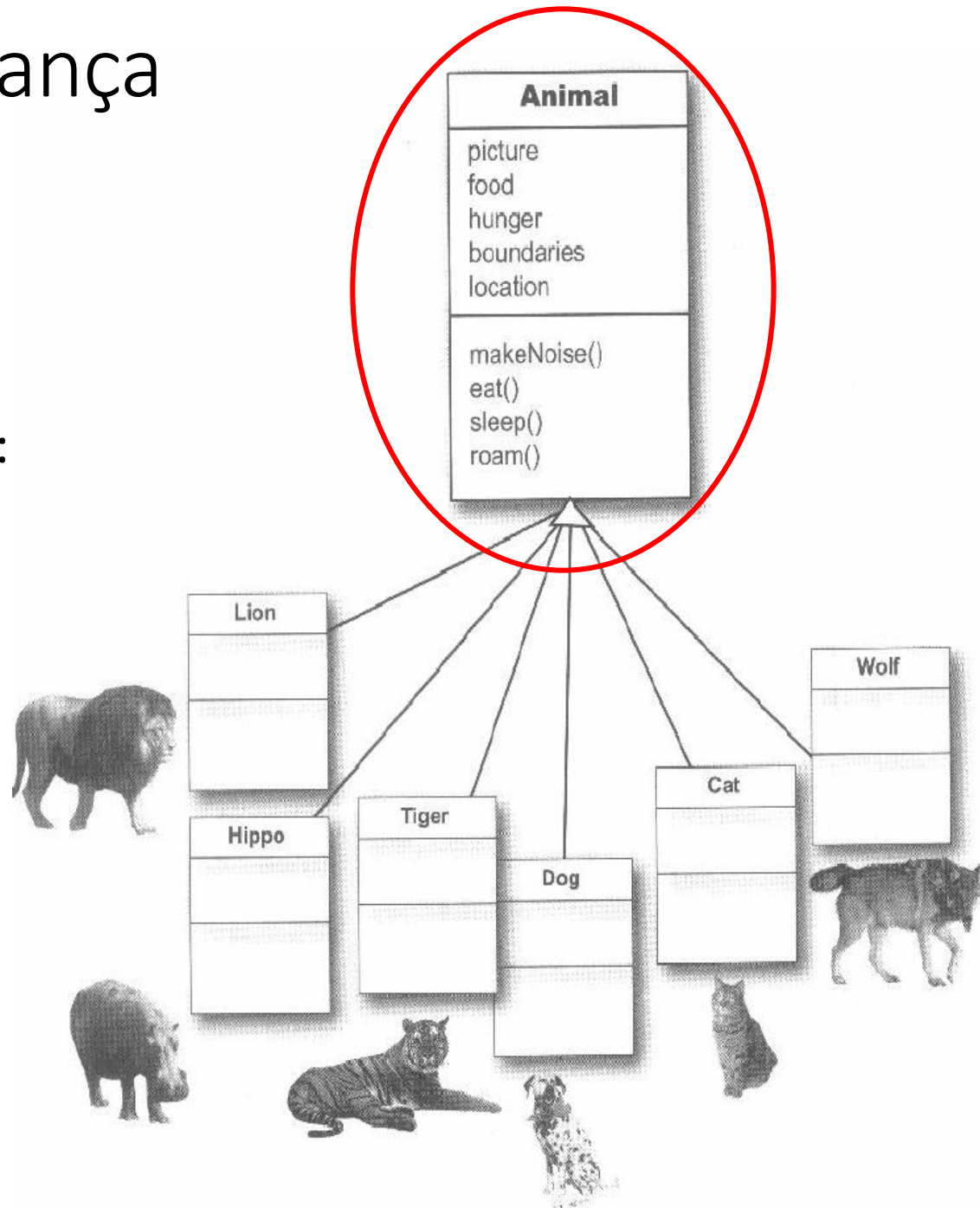




Herança

Passos para definir o relacionamento de Herança:

- 2º Passo:
- Projetar a classe que representa o estado e comportamento em comum.
- Superclasse.

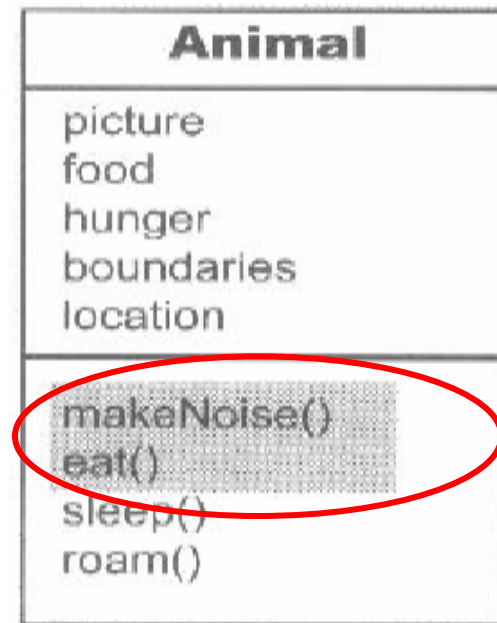




Herança

Passos para definir o relacionamento de Herança:

- 3º Passo:
 - Decidir se a subclasse necessita de comportamentos (métodos) que são específicos do tipo particular da subclasse.

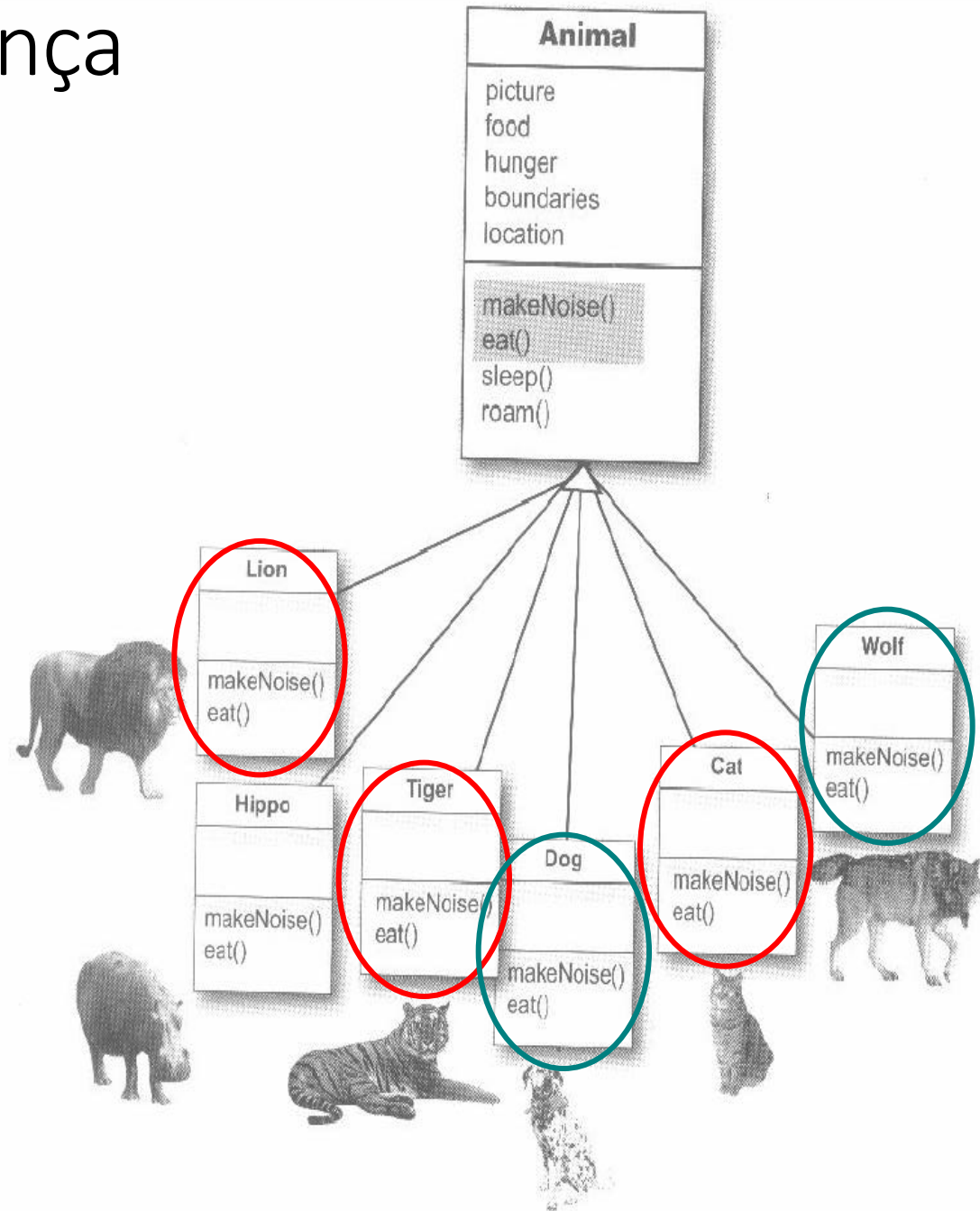




Herança

Passos para definir o relacionamento de Herança:

- 4º Passo:
- Procurar mais oportunidades para se usar a abstração,
- Ou seja, encontrar duas ou mais classes que tenham comportamento em comum

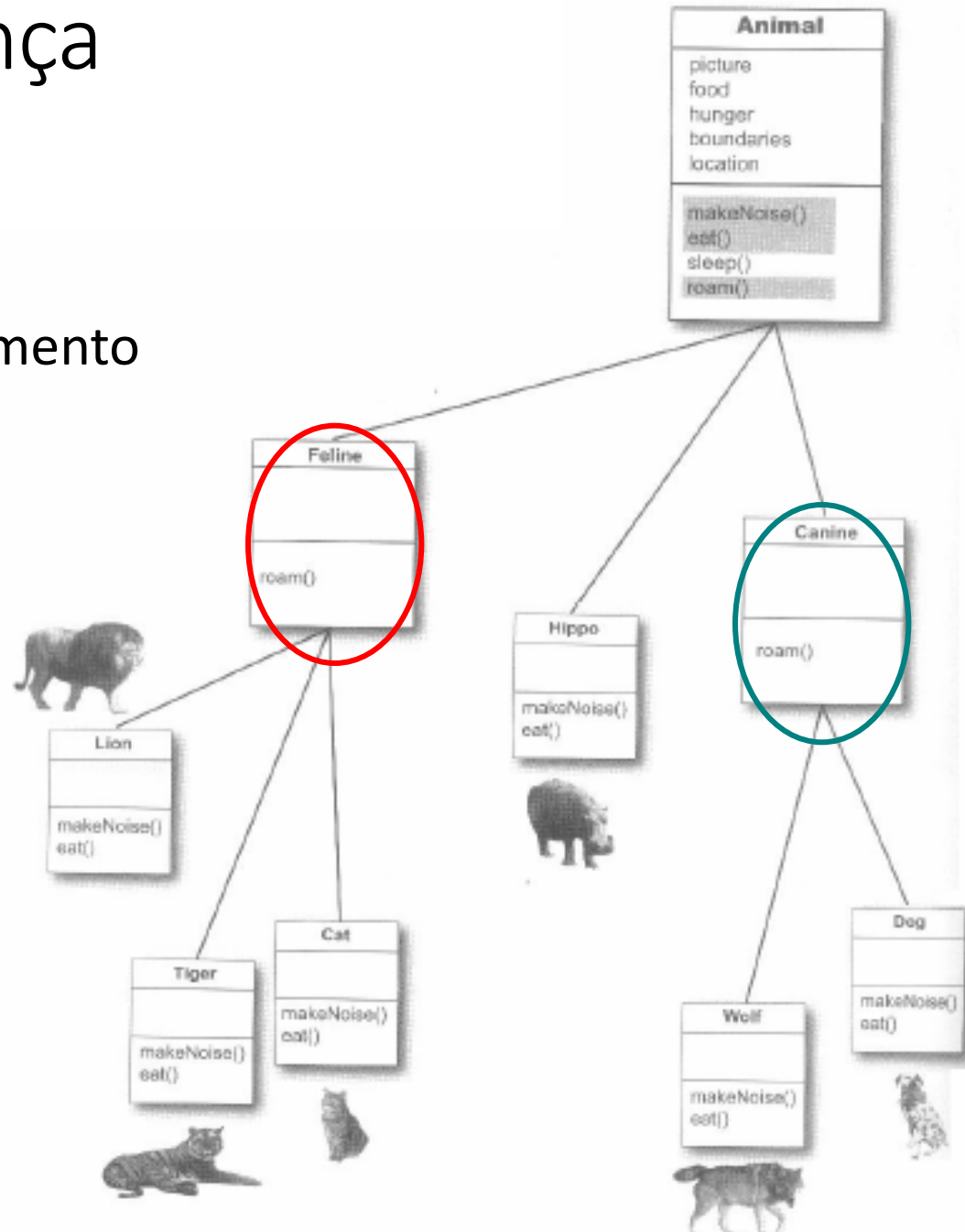




Herança

Passos para definir o relacionamento de Herança:

- 5º Passo:
 - Finalizar a Hierarquia de classes





Herança

Como funciona tudo no final então...

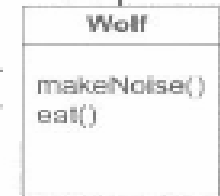
```
Wolf w = new Wolf
```

```
w.makeNoise();
```

```
w.roam();
```

```
w.eat();
```

```
w.sleep();
```





UNIVERSIDADE
VILA VELHA
ESPÍRITO SANTO

Herança em Java

Blz,

- E como eu posso escrever essas ideias??



Hierarquias de Herança

Existem duas formas comuns de hierarquias de classe:

- **Árvore:**
 - Todas as classes são parte de uma única grande hierarquia de classe.
 - Assim, há uma classe que é o antepassado original de todas classes restantes.
 - Smalltalk, Java e Object Pascal seguem esta linha.
- **Florestas:**
 - As classes estão colocadas somente nas hierarquias se tiverem um relacionamento
 - Resultando em uma floresta de muitas hierarquias pequenas, mas nenhum antepassado.
 - C++, Objective-C e Apple Object Pascal seguem esta linha.



Herança em Java

Java adota o modelo de árvore onde a classe Object é a raiz da hierarquia de classes à qual todas as classes existentes pertencem

- Toda classe tem Object como superclasse.

Mesmo quando não declaramos que uma classe herda outra, ela, implicitamente, herda Object

Mas como é que fica a sintaxe disso então...



Herança em Java

Sintaxe em Java:

```
class Subclasse extends Superclasse {  
}
```

Semântica:

- Podemos declarar um membro que, embora não seja acessível por outras classes, é herdado por suas sub-classes
 - Para isso usamos o modificador de controle de acesso *PROTECTED*
- A visibilidade na herança seria:
 - private:
 - Membros são vistos só pela própria classe e não são herdados por nenhuma outra
 - protected:
 - Membros são vistos pelas classes do pacote e herdados por qualquer outra classe
 - public:
 - membros são vistos e herdados por qualquer classe





Herança em Java

Herança Simples: Qual o resultado da execução do código abaixo?

```
1 class Funcionario {
2     String nome;
3     String cpf;
4     double salario;
5
6     // métodos devem vir aqui
7 }
8
9 /*
10  * A nomenclatura utilizada é que Funcionario é a Superclasse de Gerente, e
11  * Gerente é a Subclasse de Funcionario.
12  * Dizemos também que todo Gerente é um Funcionário.
13  */
14 class Gerente extends Funcionario {
15     int senha;
16
17     public boolean autentica(int senha){
18         if (this.senha == senha){
19             System.out.println("Acesso Permitido!");
20             return true;
21         }
22         else {
23             System.out.println("Acesso Negado!");
24             return false;
25         }
26     }
27 }
28
29 /* Todo momento que criarmos um objeto do tipo Gerente, este objeto vai possuir
30  * também os atributos definidos na classe Funcionario ,pois agora um Gerente é
31  * um Funcionario :
32  */
33 public class Banco {
34     public static void main (String [] args) {
35
36         Gerente gerente = new Gerente();
37         gerente.nome = "João da Silva";
38         gerente.cpf = "123456789";
39         gerente.salario = 1500.73;
40         gerente.senha = 4231;
41         System.out.println("Tentativa 1 - Senha Confere?: " + gerente.autentica(1235));
42         System.out.println("Tentativa 2 - Senha Confere?: " + gerente.autentica(4231));
43     } }
44 }
```



UNIVERSIDADE
VILA VELHA
ESPÍRITO SANTO

Polimorfismo

Blz,

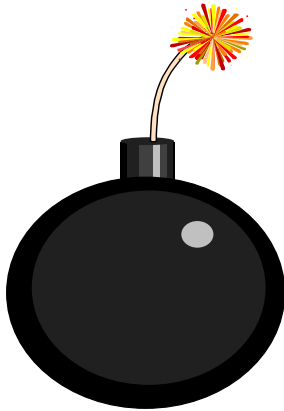
- Uma vez entendido esses conceitos...
- Vamos conversar sobre um assunto muito importante quando a gente estuda herança, que é o...



Polimorfismo

Polimorfismo, do grego: é a capacidade de assumir muitas formas.

- O Polimorfismo ocorre quando uma mesma mensagem chegando a objetos diferentes provoca respostas diferentes.



Mensagem: Fogo no pavio
Objeto 1: Bolo
Objeto 2: Bomba



- E que tipo de polimorfismo podemos ter...??



Polimorfismo

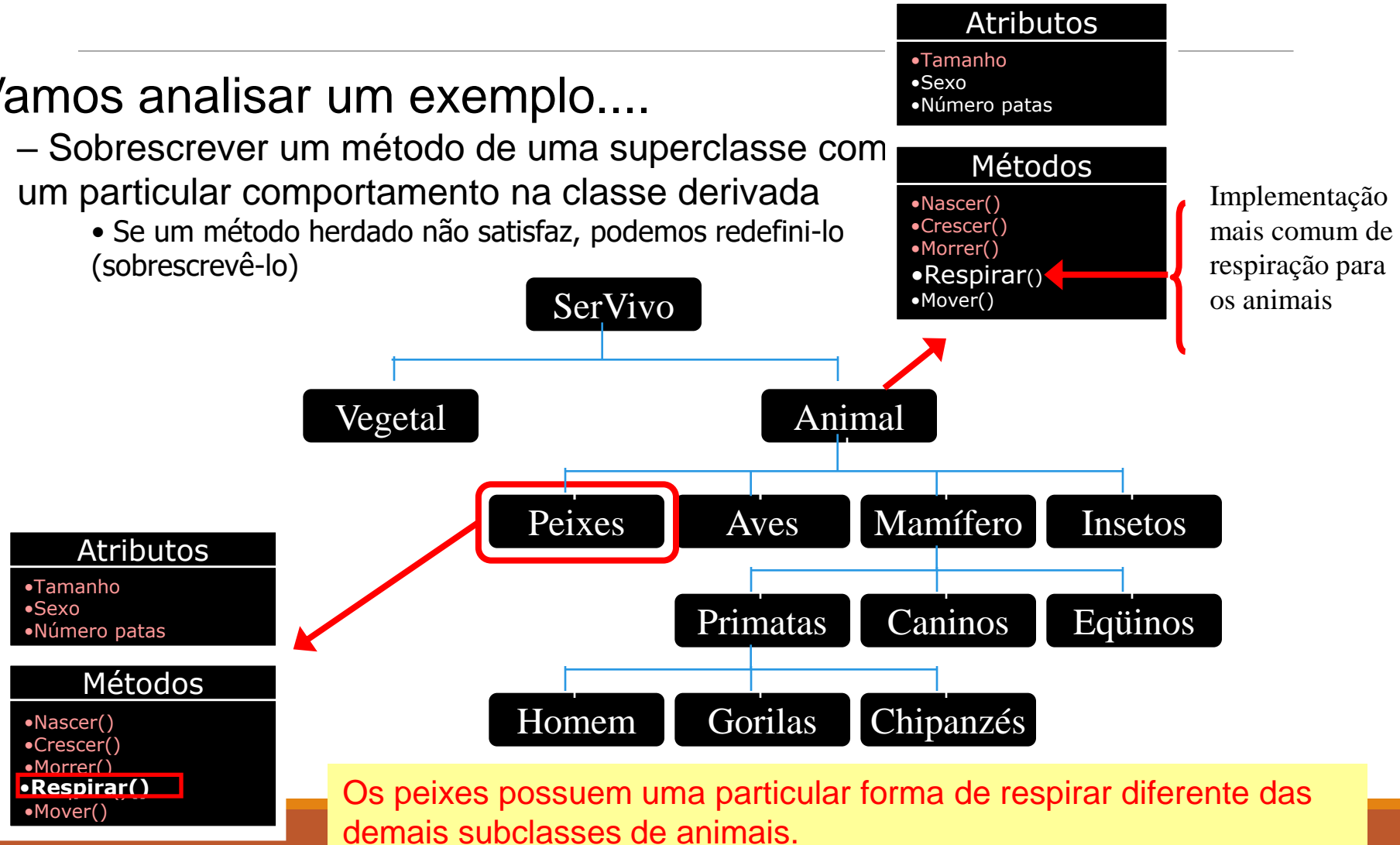
Podemos ter polimorfismo de métodos do tipo.....

- Sobrecarregando (*Overloading*)
 - Pode-se sobrecarregar um método de uma classe, criando-se assim diversas maneiras de invocá-lo.
 - Ou seja, é possível definir dois ou mais métodos com o mesmo nome, porém com assinaturas diferentes;
 - Importante: o método deve ter assinatura diferente pode ser: número ou tipos de parâmetros diferentes.
- Sobrescrevendo (*Overriding*)
 - Pode-se sobrescrever um método de mesmo nome numa classe derivada.
 - Uma classe filha pode fornecer uma outra implementação para um método herdado, caracterizando uma redefinição do método
 - Esta construção é utilizada quando a classe derivada age diferente de sua classe pai.
 - Importante: o método deve ter a mesma assinatura (nome, argumentos), senão não se trata de uma redefinição e sim, de sobrecarga.



Sobrescrevendo Métodos

- Vamos analisar um exemplo....
 - Sobrescrever um método de uma superclasse com um particular comportamento na classe derivada
 - Se um método herdado não satisfaz, podemos redefini-lo (sobrescrevê-lo)





Sobrescrita x Sobrecarga

Cuidado para não confundir.....:

```
class Forma {  
    public void aumentar(int t) {  
        System.out.println("Forma.aumentar()");  
    }  
}  
  
class Linha extends Forma {  
    // Foi feita sobrecarga e não sobrescrita!  
    public void aumentar(double t) {  
        System.out.println("Linha.aumentar()");  
    }  
}
```



Herança em Java - Final

O identificador *final* além de:

- Poder ser utilizado para definir constantes
- É utilizado para identificar uma classe que não pode ser subclassificada.
 - Por consequência, todos os métodos de uma classe final são automaticamente finais.

A palavra chave final de Java tem significados de “Isto não pode ser alterado.”

```
class Telefone { }
```

```
final class TelefoneCelular extends Telefone  
{ }
```

```
// Erro: TelefoneCelular é final!
```

```
class TelefoneAtomico extends TelefoneCelular  
{ }
```

Classe que não
pode ser
subclassificada



UNIVERSIDADE
VILA VELHA
ESPÍRITO SANTO

Herança em Java

Blz,

- Pra fechar... vamos olhar com mais atenção para a classe Object.....



Polimorfismo com java.lang.Object

O Polimorfismo é amplamente utilizado em várias as classes escritas em Java para definir os métodos:

- clone():
 - cria uma cópia do objeto (uso avançado);
- equals(Object o):
 - verifica se objetos são iguais;
- finalize():
 - chamado pelo GC (não é garantia);
- getClass():
 - retorna a classe do objeto;
- hashCode():
 - função hash;
- notify(), notifyAll() e wait():
 - para uso com threads;
- toString():
 - converte o objeto para uma representação como String.



Em especial, podemos destacar o uso do método `toString()`

- Retorna uma representação em `String` do objeto em questão;
- Permite polimorfismo em grande escala:
 - Se quisermos imprimir um objeto de qualquer classe, ele será chamado;
 - Se quisermos concatenar um objeto de qualquer classe com uma `String`, ele será chamado.



Polimorfismo com java.lang.Object

Qual o resultado....

```
class Valor {  
    int i;  
    public Valor(int i) { this.i = i; }  
}  
  
public class Teste {  
    public static void main(String[] args) {  
        Integer m = new Integer(100);  
        System.out.println(m);    // 100  
  
        Valor v = new Valor(100);  
        System.out.println(v);    // Valor@82ba41  
    }  
}
```

```
class Valor {  
    int i;  
    public Valor(int i) { this.i = i; }  
    public String toString() { return "" + i; }  
}  
  
public class Teste {  
    public static void main(String[] args) {  
        Integer m = new Integer(100);  
        System.out.println(m);    // 100  
  
        Valor v = new Valor(100);  
        System.out.println(v);    // 100  
    }  
}
```



Exercício Junto....

Blz, entendemos muita coisa hoje.....

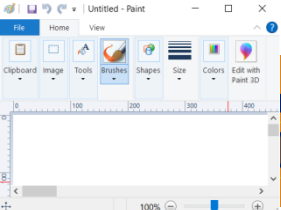
- Vamos agora construir juntos um exemplo que tenha:
 - Herança
 - Sobreescrita
 - Sobrecarga
 - Modificadores protegidos
 - toString()
- Baseado no seguinte diagrama....



Exercício Junto....

Exercício de Herança Simples

- Implementar a hierarquia para *Mamífero*
- Criar uma instância de *Homem*
- Acessar o método *Respirar()*
- Criar uma instância de *Peixe*
- Acessar o método *Respirar()* sobreescrito



Run

LA VELHA

ESPÍRITO SANTO

Exercício Junto....

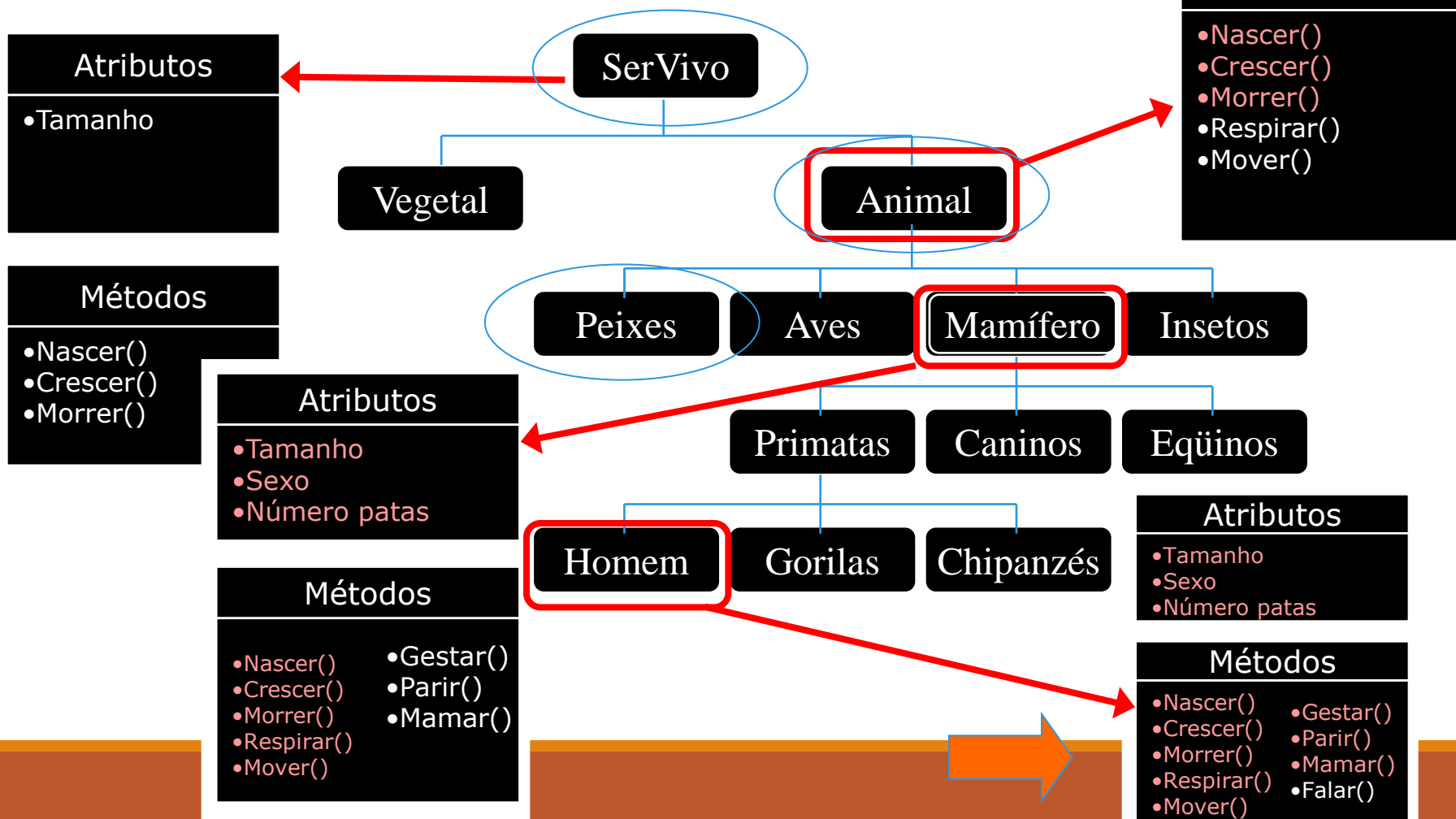
Atributos

- Tamanho
- Sexo
- Número patas

Métodos

- Nascer()
- Crescer()
- Morrer()
- Respirar()
- Mover()

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

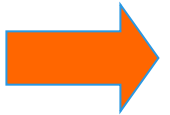




Exercício Junto....

Exercício de Sobreescrita

- Dado o modelo anterior
 - Modifique o seu código para que o método Respirar possa ser sobreescrito
 - O conteúdo do método é apenas um SOP dizendo...
 - “Respiração como um peixe”
- Lembre-se que
 - Métodos sobrescritos devem possuir a mesma assinatura!



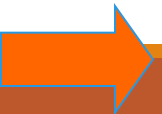


Exercícios

Blz... Agora é hora de exercitar.....

Tente resolver 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!!!

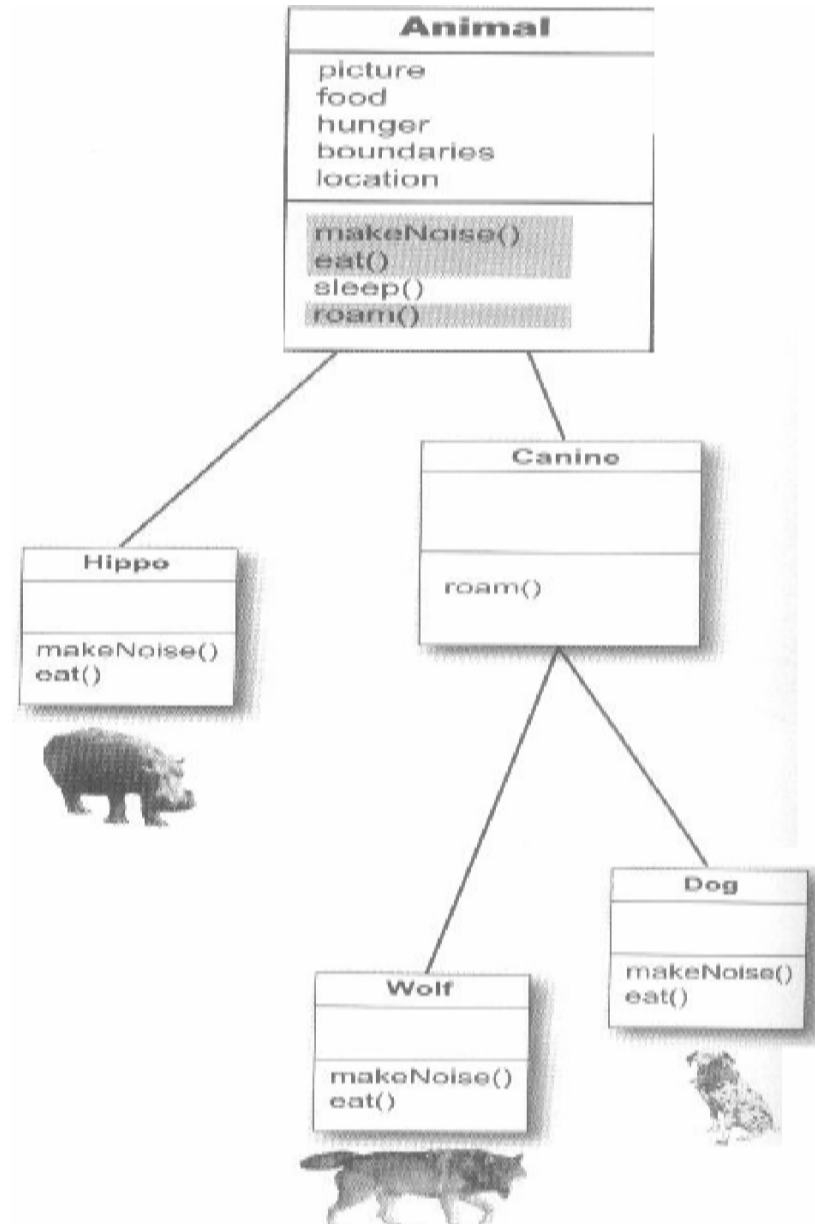




Exercício

Implementar a hierarquia ao lado

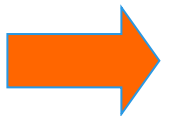
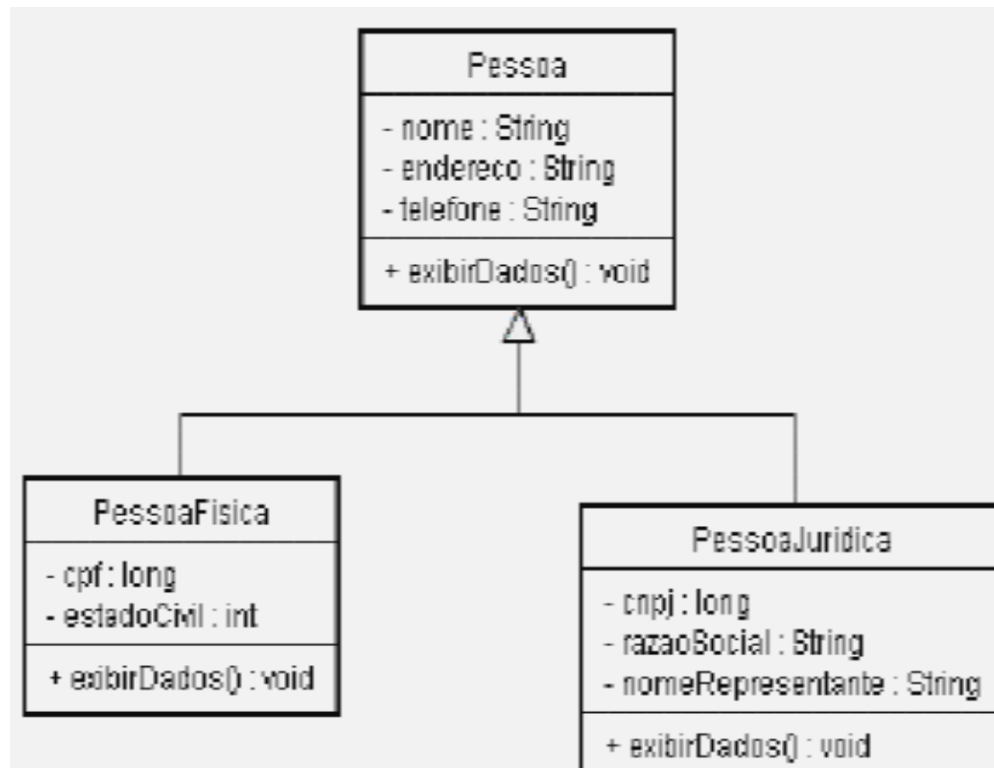
- Criar um hipopótamo;
- Criar um cachorro;
- Criar um lobo;
- Coloca os 3 para fazer barulho;
- Coloca os 3 para comer;
- Colocar os 3 para dormir.





Exercício

Implemente a hierarquia abaixo... E depois crie um simples programa e teste a implementação.

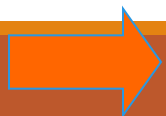




Exercício

Aproveitando o relacionamento anterior e os conceitos de herança, crie:

- a) Uma classe chamada `PesoIdealPessoa`.
 - Esta classe deve ter um método chamado `getPesoIdeal`, que recebe por parâmetro a altura da pessoa.
 - O parâmetro é do tipo `double` e deve receber a altura no formato “1.98” e 1.70” por exemplo.
- b) Criar duas classes herdando desta, com os nomes `PesoIdealHomem`, e `PesoIdealMulher`.
 - Os cálculos para peso ideal de homem e mulher são os seguintes:
 - Para homens = $(72.7 * altura) - 58$;
 - Para mulheres = $(62.1 * altura) - 44.7$;
- c) Criar um aplicativo (classe `PesoIdealPrincipal`) que recebe via Scanner o sexo (com as opções “M” ou “F” para Masculino ou Feminino) , a altura da pessoa e retorna na saída o peso ideal para a pessoa.
 - O programa principal deve criar a classe `PesoIdealHomem` ou `PesoIdealMulher` conforme o parâmetro indicado para sexo.





Exercício

Nas classes PessoaJuridica e PessoaFisica temos os seguintes atributos:

- CPF e CNPJ

Faça a sobreposição do método public boolean equals (Object o), herdado da classe Object, nas classe específicas,

- Para que ele considere que 2 objetos PessoaJuridica ou PessoaFisica são iguais quando eles tiverem o mesmo cnpj ou cpf respectivamente.

Faça teste com e sem essa sobreposição

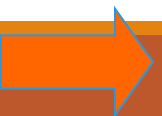


Exercício

Dada a classe `Circulo`, implemente uma nova classe `CirculoColorido` utilizando herança.

Esta nova classe apresenta o mesmo comportamento de `Circulo`, mas possui a capacidade adicional de manter a informação da cor de desenho do traçado do círculo e uma cor interna para o preenchimento da figura geométrica.

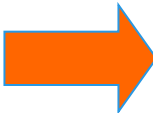
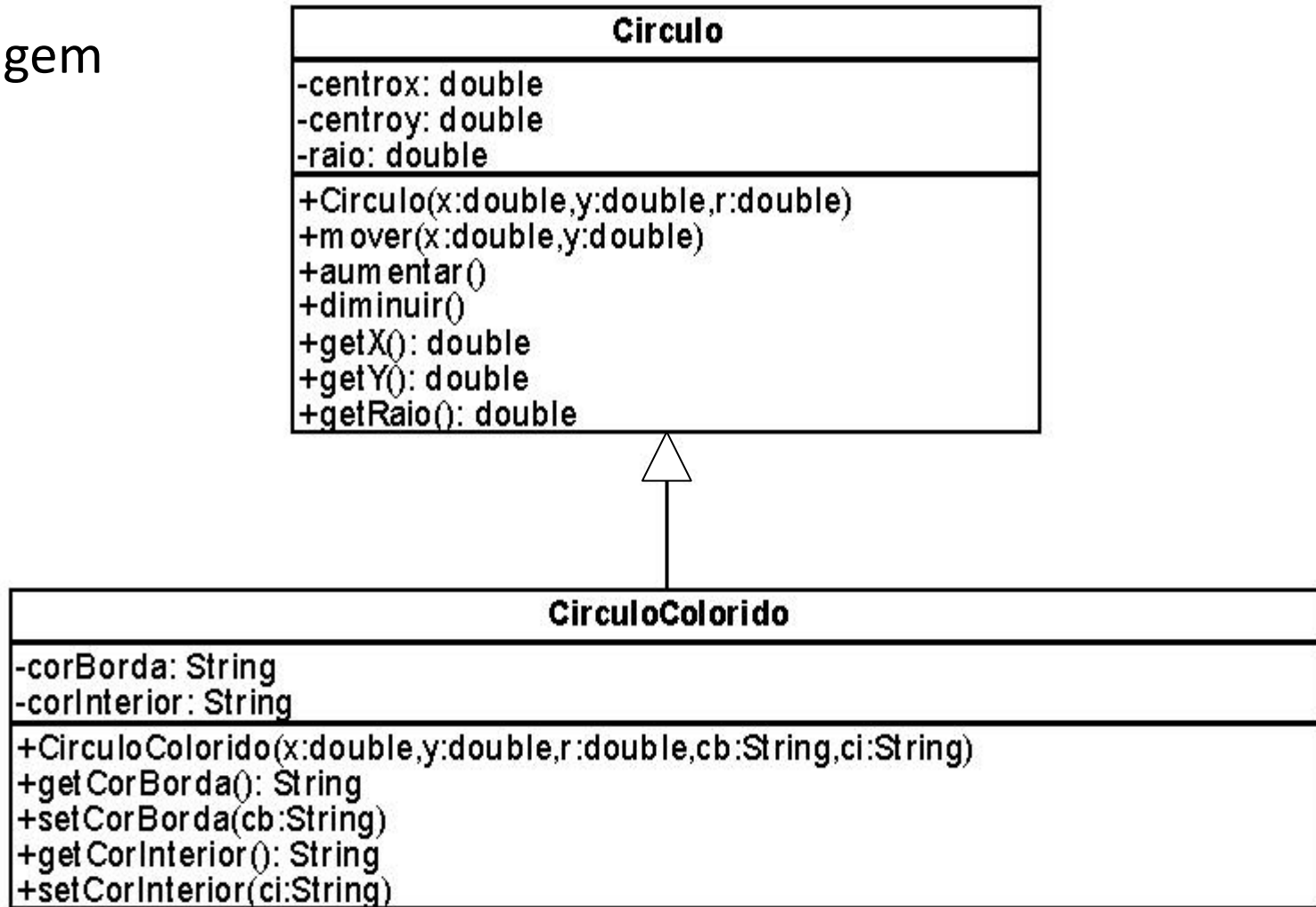
A modelagem e o código original da classe `Circulo` são apresentados a seguir para ajudá-lo na tarefa.





Exercício

Modelagem





Exercício

Código

```
1 public class Circulo {
2     private double centrox;
3     private double centroy;
4     private double raio;
5     public Circulo(double x, double y, double r) {
6         if (x >= 0) centrox = x;
7         else centrox = 0;
8         if (y >= 0) centroy = y;
9         else centroy = 0;
10        if (r > 0) raio = r;
11        else raio = 1;
12    }
13    public void mover(double x, double y) {
14        centrox = x;
15        centroy = y;
16    }
17    public void aumentar() {
18        raio++;
19    }
20    public void diminuir() {
21        raio--;
22    }
23    public double getX() {
24        return centrox;
25    }
26    public double getY() {
27        return centroy;
28    }
29    public double getRaio() {
30        return raio;
31    }
32 }
```