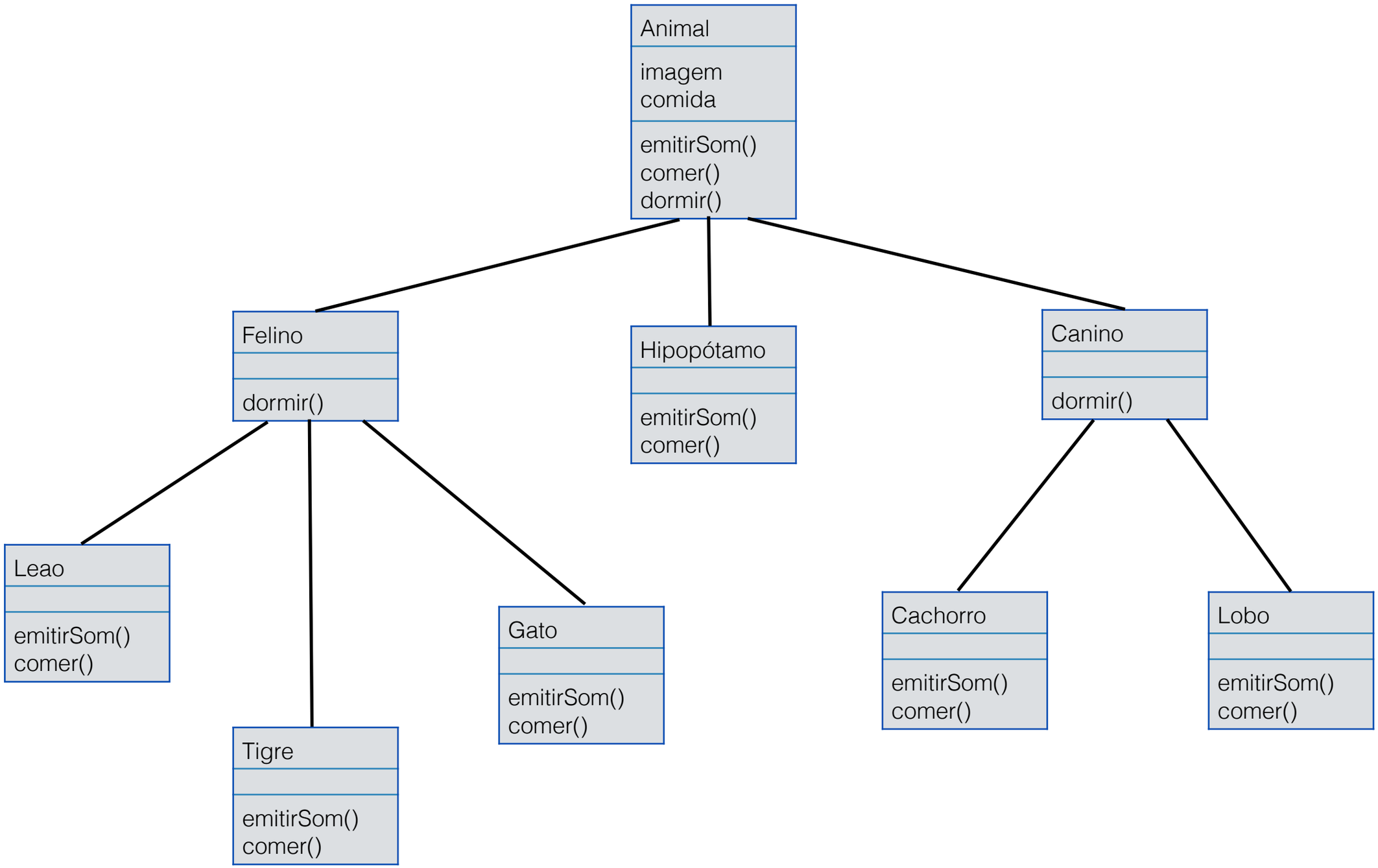

JAVA™

Interfaces

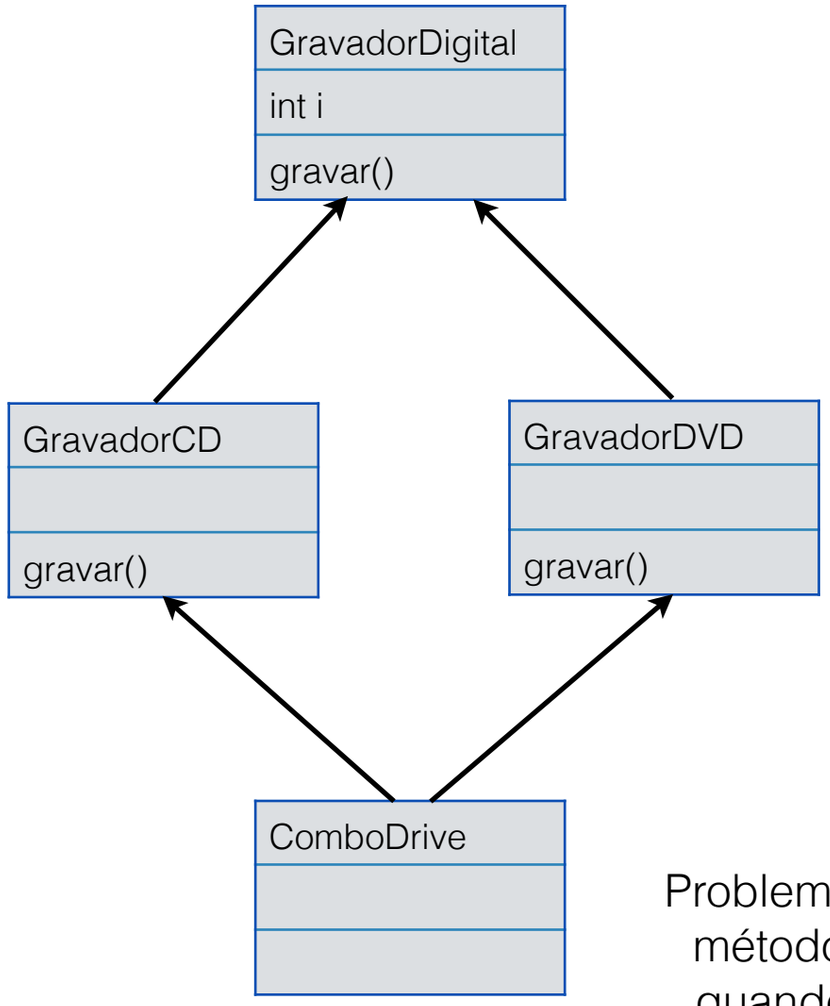

Animal
imagem
comida
emitirSom()
comer()
dormir()
Felino
dormir()
Hipopótamo
emitirSom()
comer()
Canino
dormir()
Leao
emitirSom()
comer()
Tigre
emitirSom()
comer()
Gato
emitirSom()
comer()
Cachorro
emitirSom()
comer()
Lobo
emitirSom()
comer()

Interfaces

Mas há um problema na abordagem de “duas superclasses”...

Chama-se “herança múltipla” e pode ser algo realmente perigoso.

Tanto GravadorCD quanto GravadorDVD herdam de GravadorDigital e as duas classes sobrepõem o método gravar(). Ambas herdam a variável de instância “i”.



Imagine se a variável de instância “i” for usada por GravadorCD e GravadorDVD, com valores diferentes. O que aconteceria se ComboDrive precisasse usar os dois valores de “i”?

Problema da herança múltipla. Que método Gravar() será executado quando chamarmos gravar() em ComboDrive?

Interfaces

Opção um

Tomaremos o caminho mais fácil e vamos inserir os métodos de Pet na classe Animal.

Opção dois

Começaremos com a opção um como base, inserindo os métodos de Pet na classe Animal, e teremos de "foçar" as subclasses de Animal a sobrepô-los.

Opção três

Inserir os métodos Pet SOMENTE nas classes a que pertencem.

Interfaces

Do que nós realmente precisamos?

Uma maneira de termos comportamentos de animal doméstico apenas em classes Pet

Uma maneira de garantir que todas as classes Pet tenham os mesmos métodos definidos (mesmo nome, mesmos argumentos, mesmos tipos de retorno, nenhum método faltando, etc.), sem ser preciso cruzar os dedos e rezar para que todos os programadores pensem igual.

Uma maneira de nos beneficiarmos do polimorfismo para que todos os objetos Pet possam ter seus métodos chamados, sem que tenhamos que usar argumentos, tipos de retorno e matrizes específicos de cada classe Pet.

Interfaces

Para explorar o polimorfismo, precisamos de interfaces (mas não do tipo GUI). Projetando e codificando com especificações de interface, poderemos ir além da simples herança.

Algumas das partes mais interessantes do Java não poderiam ao menos existir sem interfaces, portanto mesmo se você não projetar com elas, ainda terá que usá-las.

Interfaces

O que é uma interface?

É uma classe 100% abstrata.

O que é uma classe abstrata?

É uma classe que não pode ser instanciada.

Interfaces

Sabemos que podemos dizer: **Lobo l = new Lobo();**

E sabemos que podemos dizer: **Animal l = new Gato();**

Mas aqui começa a ficar estranho:

Animal a = new Animal();

Qual a aparência de um novo objeto Animal()?

Objetos Estranhos!

Interfaces

Algumas classes ***não*** deviam ser instanciadas!

Se marcarmos uma classe com abstract, o compilador impedirá que qualquer código crie uma instância dessa classe.

```
abstract class Canino extends Animal{
    public void dormir(){}
}
```

Quando projetar a estrutura das classes, terá que decidir que classes serão abstratas e quais serão concretas. As classes concretas são aquelas específicas o suficiente para serem instanciadas. Uma classe concreta significa que não há problemas em se criar objetos desse tipo.

Interfaces

Métodos Abstratos

Um método abstrato não tem corpo!

```
public abstract void comer();
```

Não há implementação no método. Apenas termine-o com um ponto e vírgula.

Se declarar um método como abstrato, também DEVERÁ marcar a classe como abstrata. Não é possível ter um método abstrato em uma classe não abstrata.

Interfaces

O que define a classe Cachorro?

As variáveis de instância e os métodos implementados.

Mas não é só isso!

Tudo que existir na classe Canino fará parte de Cachorro.

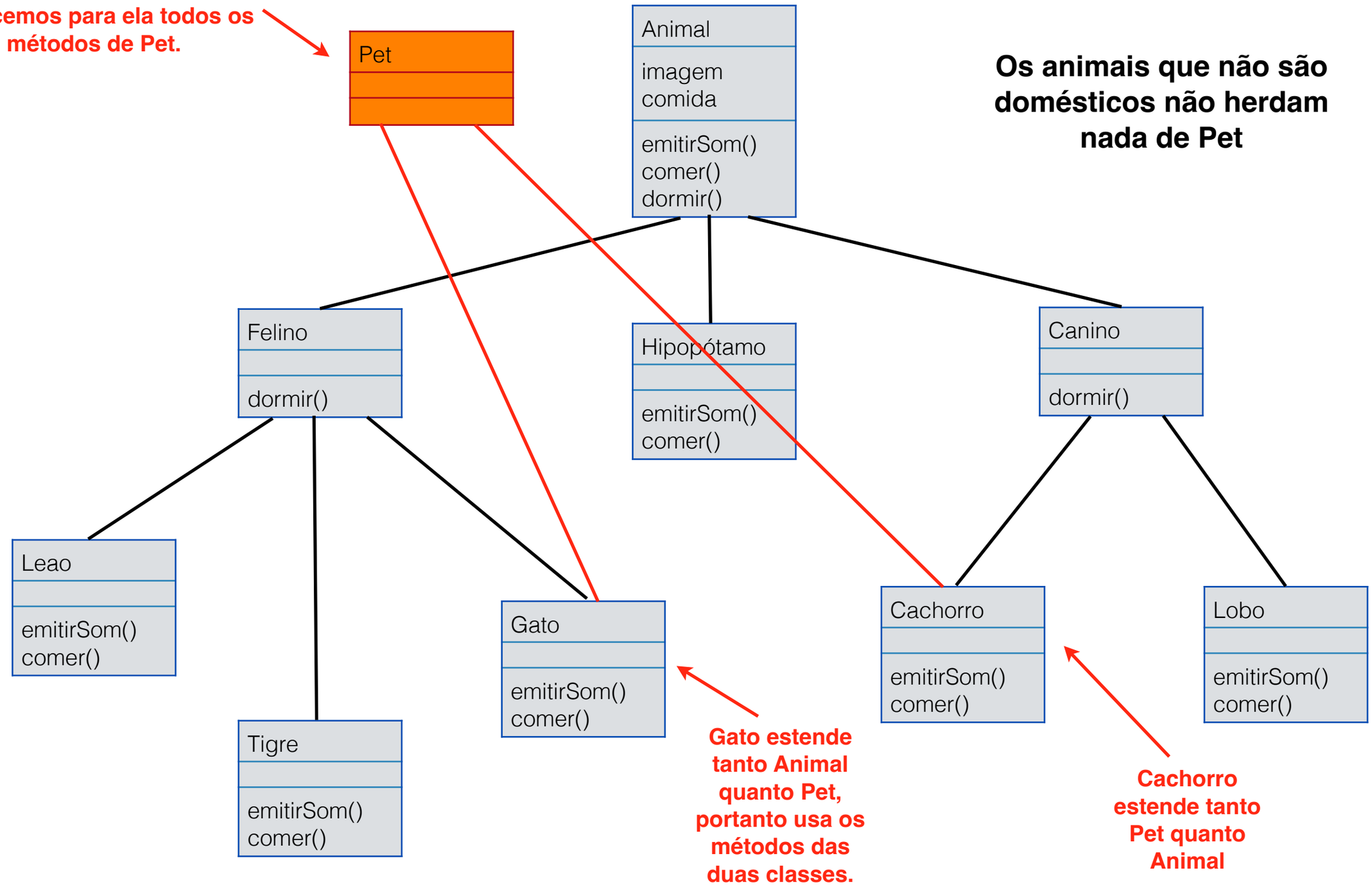
Tudo que existir na classe Animal fará parte de Cachorro.

De acordo com o teste É-UM, cachorro é todas essas coisas -Canino e Animal

Com isso em mente, pense: como podemos usar este diagrama de classes num programa de PetShop? Nossas classes não têm nenhum comportamento de animal doméstico. Um objeto Pet precisa de métodos como serAmigavel() e brincar().

Interfaces

Criamos uma nova superclasse abstrata chamada Pet e fornecemos para ela todos os métodos de Pet.



Os animais que não são domésticos não herdam nada de Pet

Gato estende tanto Animal quanto Pet, portanto usa os métodos das duas classes.

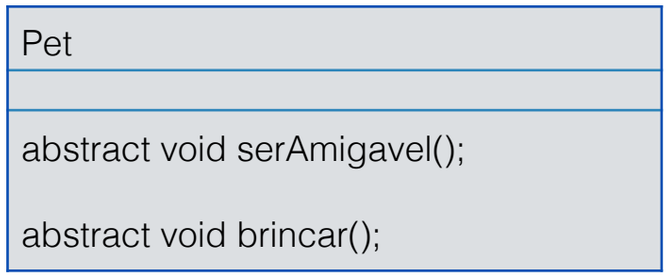
Cachorro estende tanto Pet quanto Animal

Interfaces

O Java tem uma solução. A interface.

Uma interface Java é como uma classe 100% abstrata.

Todos os métodos de uma interface são abstratos, portanto, qualquer classe que FOR-UM animal doméstico DEVE implementar (isto é, sobrepor) os métodos de Pet.



Para definir uma interface:

public interface Pet{...} ← **Use a palavra chave "interface" em vez de "class"**

Interfaces

O Java tem uma solução. A interface.

Para implementar uma interface:

public class Cachorro extends Canino implements Pet{...}

Use a palavra chave “implements” seguida do nome da interface. Observe que, quando você implementar uma interface, ainda poderá estender uma classe.

Interfaces

Criando e implementando a interface Pet:

```
public interface Pet{
    public abstract void serAmigavel();
    public abstract void brincar();
}
```

Todos os métodos da interface são abstratos, portanto, DEVEM terminar com ponto-e-vírgula. Lembre-se de que eles não têm corpo!

Cachorro É-UM animal e Cachorro É-UM animal doméstico

```
public class Cachorro extends Canino implements Pet{
    public abstract void serAmigavel(){...}
    public abstract void brincar(){...}

    public void emitirSom(){...}
    public void correr(){...}
}
```

Insira “implements” seguido do nome da interface

Foi declarado um objeto Pet, portanto DEVEM ser implementados os métodos de Pet. É seu contrato. Observe as chaves em vez do ponto-e-vírgula.

Esses são apenas métodos de sobreposição comuns.

Interfaces

Como saber se deve-se criar uma classe, uma subclasse, uma classe abstrata ou uma interface?

Crie uma classe que não estenda nada quando sua nova classe não passar no teste É-UM com nenhum outro tipo.

Crie uma subclasse (em outras palavras, estenda uma classe) somente quando tiver que criar uma versão ***mais específica*** de uma classe e precisar sobrepor ou adicionar novos comportamentos.

Use uma classe abstrata quando quiser definir um ***modelo*** para um grupo de subclasses e tiver pelo menos algum código de implementação que todas as subclasses possam usar. Torne a classe abstrata quando quiser garantir que ninguém possa criar objetos desse tipo.

Use uma interface quando quiser definir uma ***função*** que outras classes possam desempenhar, independentemente de onde essas classes estejam na árvore de herança.