Objetos, herança e polimorfismo

Maurício Linhares

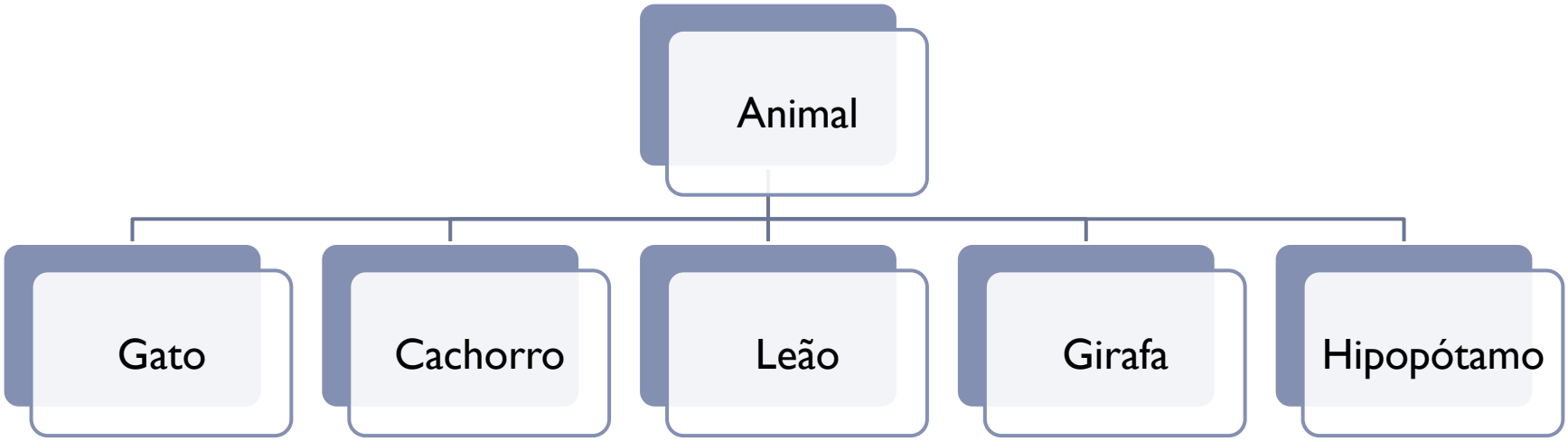
O que é a herança?

- É uma das propriedades básicas da orientação a objetos, aonde você define que uma classe herda os atributos e ações de uma outra classe;

- Funciona como uma forma de evitar a repetição de código, em vez de espalhar o código em diversas classes, tudo o que é genérico fica em uma classe pai e todas as classes filhas herdam o código;

- Todas as classes em Java herdam da classe pai chamada Object (o que quer dizer que todo mundo é um Object);

Um exemplo de herança



Os animais

- Todos os nossos animais herdam da classe Animal, porque todos eles fazem coisas parecidas, como comer, andar, dormir e fazer barulhos;

- Além dessas ações, a classe animal tem propriedades para designar peso, altura e a cor de sua pele (ou pelos);

Classe Animal

```
public class Animal {

    private int peso;
    private int altura;
    private String cor;

    public void comer() {
            System.out.println("Sou onívoro, como tudo!");
    }

    public void dormir() {  System.out.println("Deitei e dormi");          }

    public void andar() { System.out.println("Estou andando por aí"); }

    public void fazerBarulho() { System.out.println("Huf! Huf! Huf!"); }

}
```

Indo para os outros animais

- Agora nós não precisamos mais implementar essas mesmas funcionalidades básicas nos nossos animais, todos eles vão herdar as qualidades da classe Animal;
- Quando um dos nossos animais não estiver interessado na funcionalidade provida pela classe Animal ele vai poder alterar essa funcionalidade só para ele;
- Os objetos que herdam de animal são versões especializadas do nosso animal;
- Para informar que uma classe herda da outra nós usamos a palavra reservada "**extends**";

Exemplo de animal especializado

```
public class Hipopotamo extends Animal {

    public void comer() {
        System.out.println("Comendo plantas");
    }

}
```

O método comer foi sobrescrito

Usando um hipopótamo

```
public class HipopotamoTest {

    public static void main(String[] args) {
            Animal hipopotamo = new Hipopotamo();

            hipopotamo.setCor("cinza");
            hipopotamo.setPeso(500);

            hipopotamo.comer();
            hipopotamo.andar();
            hipopotamo.dormir();

            System.out.println(
            "O meu hipopótamo pesa " + hipopotamo.getPeso() +
            " e é da cor " + hipopotamo.getCor());
    }

}
```

O que aconteceu?

- Nós podemos chamar, no hipopótamo, todos os métodos que haviam sido definidos em animal, porque hipopótamo herda de Animal (ele é um animal);

- A implementação utilizada é a que está na classe Animal, a não ser no método "**comer()**", como a classe Hipopotamo sobrescreveu o método, agora a versão do método que é chamada é a do próprio Hipopotamo, e não mais a de Animal;

O que acontece com esse código?

```
public class AnimalTest {

   public static void main(String[] args) {

         Animal animal = new Animal();
         animal.comer();
         animal.andar();
         animal.dormir();

   }

}
```

Vejamos como seria implementado um leão

```
public class Leao extends Animal {

    private boolean chefe;

    public void setChefe(boolean chefe) { this.chefe = chefe; }

    public boolean isChefe() { return chefe;    }

    public void comer() {
            System.out.println("Comendo carne!");
    }

    public void fazerBarulho() {
            System.out.println("Roooooooooaaaaaaaaaaarrrrrrrrrrr");
    }

}
```

O que o leão tem?

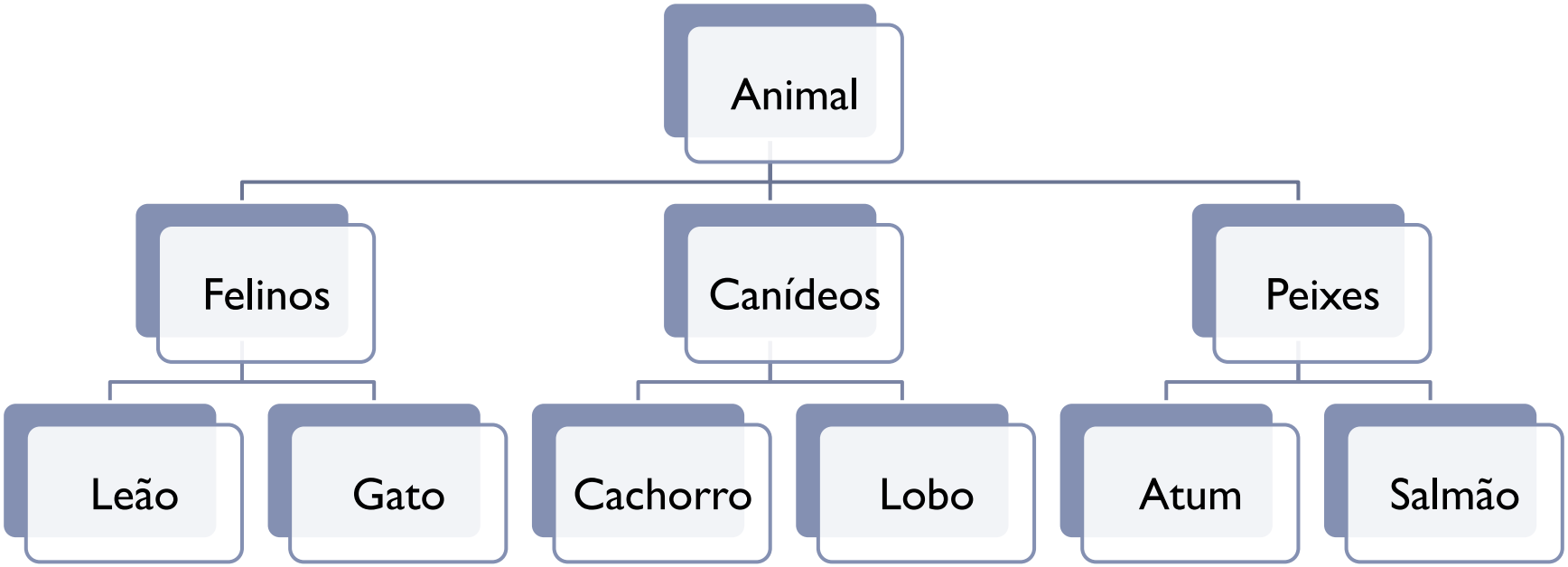
- O leão sobrescreve dois métodos de Animal e ainda adiciona uma nova propriedade, a "chefe", para indicar se esse leão é o chefe do bando;

- Ao criar uma nova propriedade e sobrescrever os seus métodos, a classe Leao especializa os comportamentos definidos originalmente na classe Animal, mas continua podendo ser tratado como um animal;

Tratando um leão como um Animal

```
public class LeaoTest {

   public static void main(String[] args) {
         Animal animal  = new Leao();

         animal.andar();
         animal.comer();
         animal.fazerBarulho();

   }

}
```

Organizando melhor a nossa árvore de objetos



Sobrescrita de métodos

- A sobrescrita de método acontece quando uma subclasse redefine um método encontrado originalmente em uma de suas superclasses;

- Para sobrescrever um método é necessário redefini-lo com o mesmo nome, os mesmos parâmetros e o mesmo tipo de retorno (mesma assinatura);

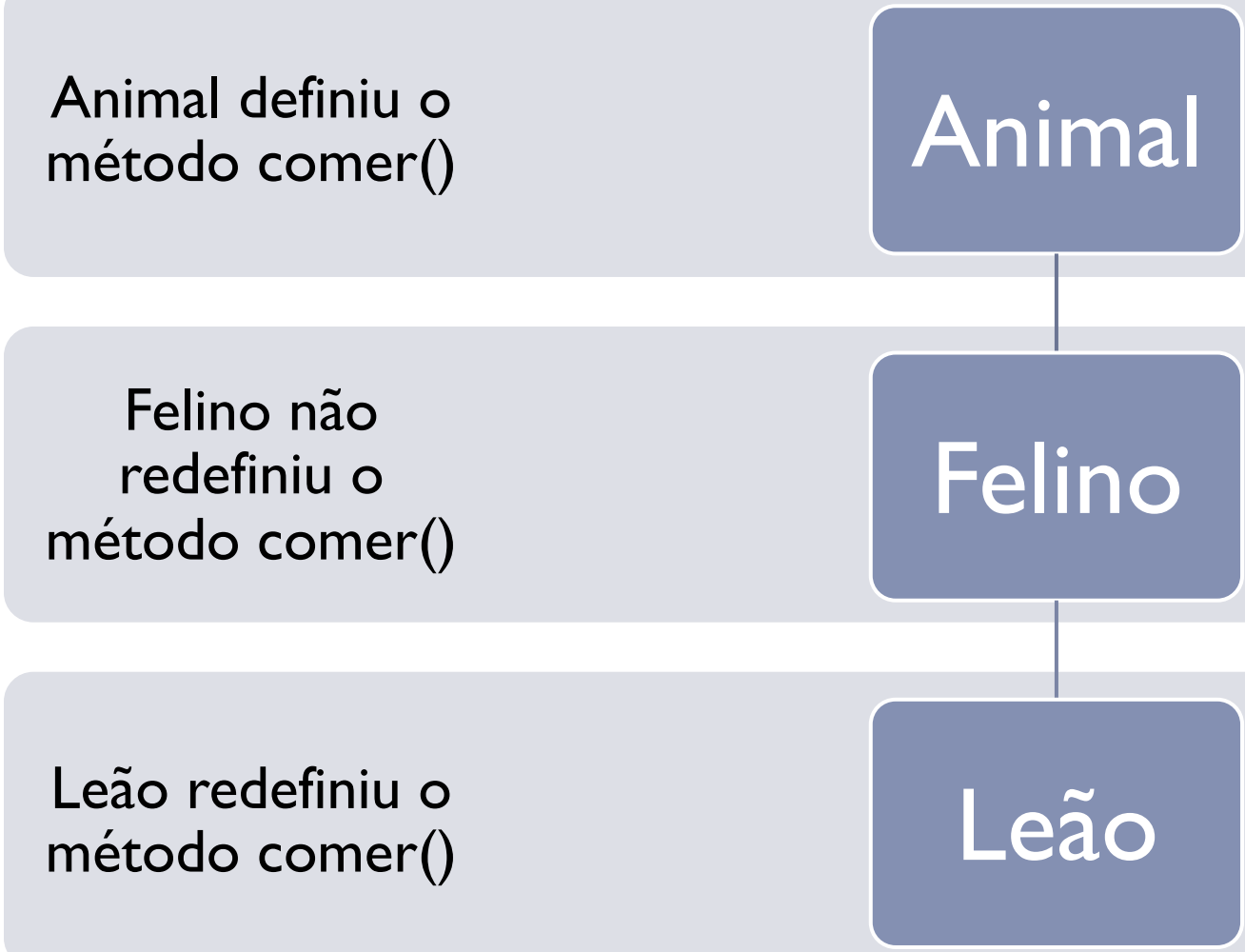
- Quando a máquina virtual vai escolher um método a ser chamado, ela procura sempre pelo mais especializado;

Despacho de métodos

Animal definiu o método comer()

Felino não redefiniu o método comer()

Leão redefiniu o método comer()



Exemplo de despacho de métodos

```
public class FelinoTest {

   public static void main(String[] args) {
         Animal animal  = new Animal();
         animal.comer();
         Felino felino = new Felino();
         felino.comer();
         Leao leao = new Leao();
         leao.comer();
   }

}
```

Sobrecarga de métodos

- Acontece quando se define um método com o mesmo nome de um método da classe atual ou de uma superclasse, mas os parâmetros são alterados;

- Pode ser utilizado para mudar os tipos que podem ser aceitos através de uma certa operação (como imprimir um número ou uma data);

- Um dos exemplos mais comuns está lá no objeto System.out...

Exemplo de sobrecarga

```
public class PrintTest {

   public static void main(String[] args) {
         System.out.println(10);
         System.out.println(33.89);
         System.out.println("Qualquer coisa");
         System.out.println(new Date());
   }

}
```

Definindo métodos sobrecarregados

```
public class Imprimidor {

	private PrintStream saida = System.out;

	public void imprimir( double numero ) {
		this.saida.println( numero );
	}

	public void imprimir( int numero ) {
		this.saida.println( numero );
	}

	public void imprimir( float numero ) {
		this.saida.println( numero );
	}

	public void imprimir( String texto ) {
		this.saida.println( texto );
	}

}
```

Selecionando métodos sobrecarregados

- A máquina virtual sempre escolhe o método que tiver o tipo mais próximo do passado como parâmetro na hora de imprimir;

- se você passar um objeto Date, ela vai procurar por um método que imprima Dates, só depois ela vai buscar por um método que imprima uma das subclasses;

- Um método que receba "**Object**" recebe qualquer coisa;

Classes abstratas

- Classes abstratas são classes que existem apenas com o propósito de serem extendidas, elas não podem ser utilizadas diretamente;

- Uma classe abstrata define a "base" de um certo conjunto de objetos, ela forma o elo comum entre todos eles;

- Classes abstratas em Java são definidas através do uso da palavra reservada "abstract" antes da definição da classe;

Exemplo de classe abstrata

```
public abstract class Cliente {

    private String nome;

    private String email;

    private Endereco endereco;

    private Date clienteDesde;

    //métodos get-set
}
```

Usando uma classe abstrata

- Todos os métodos e variáveis que não forem private ficam disponíveis para acesso direto pelas subclasses;

- As subclasses herdam todos os comportamentos da classe abstrata e podem sobrescrever todos os que não estejam marcados como "**final**" (Métodos marcados como final não podem ser sobrescritos);

- Mesmo que as variáveis de instância estejam privadas e invisíveis, elas podem ser acessadas pelas subclasses se houverem métodos get-set;

Uma subclasse

```
public class PessoaFisica extends Cliente {

   private Date nascidoEm;

   private String cpf;

   //métodos get-set

}
```

Outra subclasse

```
public class PessoaJuridica extends Cliente {

   private String cnpj;

   private String inscricaoEstadual;

   private String nomeDeFantasia;
   //métodos get-set
}
```

Definindo um mensageiro para clientes

```
public class Mensageiro {

   public void enviarEmailI( Cliente cliente ) {
         System.out.println(
                  "Enviando um e-mail para -> " +
                   cliente.getEmail() +
                  " do tipo " +
                  cliente.getClass().getCanonicalName() );
   }

}
```

Usando as classes abstratas

```
Mensageiro mensageiro = new Mensageiro();

PessoaFisica pessoaFisica = new PessoaFisica();
pessoaFisica.setEmail("jose@gmail.com");

PessoaJuridica pessoaJuridica = new PessoaJuridica();
pessoaJuridica.setEmail("comerciante@comercio.com");

mensageiro.enviarEmailI(pessoaFisica);
mensageiro.enviarEmailI(pessoaJuridica);
```

O que não se pode fazer

```
Mensageiro mensageiro = new Mensageiro();

Cliente cliente = new Cliente(); //não pode
cliente.setEmail("jose@gmail.com");

mensageiro.enviarEmailI(cliente);
```

Quando usar classes abstratas?

- Quando não fizer sentido ter objetos da classe base (não faz sentido ter um cliente que não é nem pessoa jurídica nem pessoa física);

- Quando você quer obrigar a criação de subclasses e apenas subclasses possam ser utilizadas;

- Quando você quer deixar a implementação de alguma coisa para as subclasses dos seus objetos, através de **métodos abstratos**;

Métodos abstratos

- São métodos que são definidos em uma classe, mas não são implementados nela, apenas em suas subclasses;
- Normalmente são criados para se definir um meio de acesso comum a uma ação que pode ser diferente para cada tipo de objeto;
- Se uma classe tem um método abstrato, é obrigatório que ela também seja marcada como abstrata;

Exemplo de método abstrato

```
public abstract class Conta {

    private double saldo;

    public abstract double getLimite();

    public double getSaldo() {
            return saldo;
    }

}
```

Uma implementação do método abstrato

```
public class Poupanca extends Conta {

    public double getLimite() {
            return this.getSaldo();
    }

}
```

Outra implementação...

```
public class ContaCorrente extends Conta {

    private double chequeEspecial;

    public double getLimite() {
            return this.chequeEspecial + this.getSaldo();
    }

    public double getChequeEspecial() { return chequeEspecial; }

    public void setChequeEspecial(double chequeEspecial) {
            this.chequeEspecial = chequeEspecial;
    }

}
```

Exercício

- Crie a classe abstrata Polígono, essa classe deve ter o método abstrato getArea;

- Crie as classes concretas Triangulo, Quadrado, Retangulo, Pentágono e Hexágono, implemente nessas classes os métodos que calculem a área total da forma geométrica de todas elas;

- Algumas implementações podem ser herdadas, evite a repetição de código;