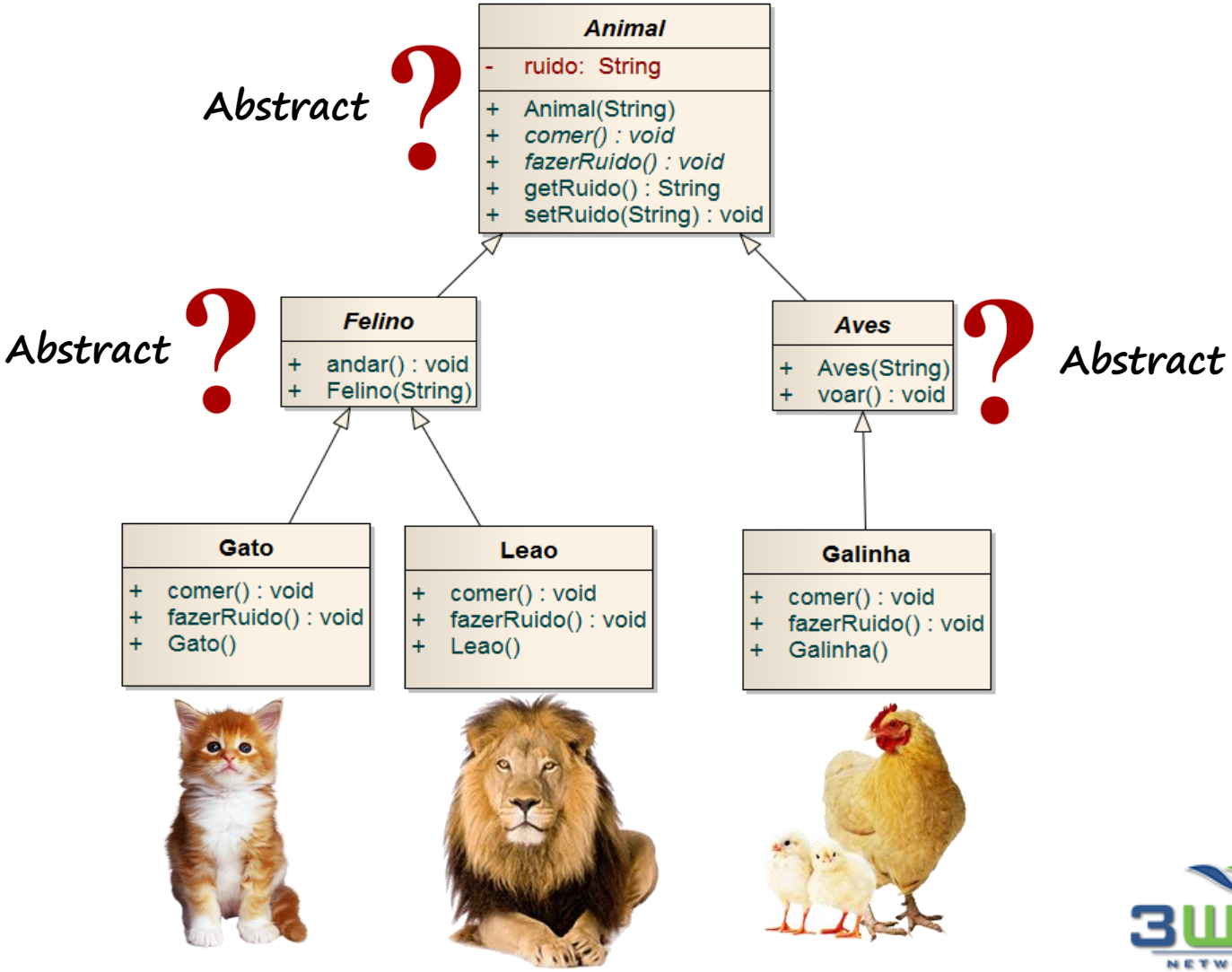
Java Orientado a Objetos
Classes Abstratas, Internas e Interfaces

Classes Abstratas



Métodos Abstratos

São métodos criados nas classes **abstratas** sem implementação.

<modificador>* abstract <tipoRetorno><nomeMetodo>(<argumento>*);

*Toda classe que contém um método **abstract** deve ser declarada **abstract**.*

Classe abstrata não precisa ter método abstrato

Exemplos de implementação

```
public abstract class Animal {
    private String ruido; // atributo da classe abstrata

    public Animal( String ruido ) { //construtor
        this.ruido = ruido;
    }
    public abstract void fazerRuido(); // métodos abstratos
    public abstract void comer();

    //get e set
    public String getRuido() { return ruido;}
    public void setRuido(String ruido) { this.ruido = ruido;}
}
```

```
public abstract class Felino extends Animal {
    public Felino( String ruido ) {
        super(ruido);
    }

    public void andar(){ System.out.println("Anda com 4 patas");}
}
```

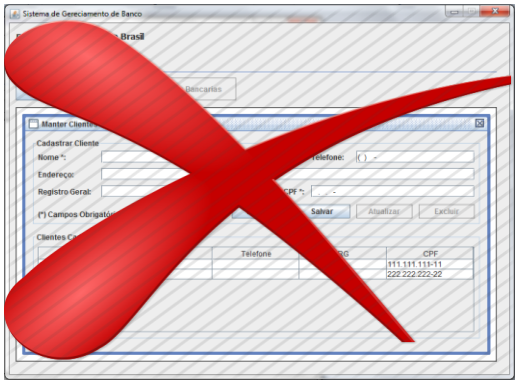
Exemplos de implementação

```
public class Gato extends Felino {

    public Gato() {
        super("Miauuuu, miauuu");
    }

    @Override
    public void fazerRuido() {
        System.out.println("Miar= " + this.getRuido());
    }

    @Override
    public void comer() {
        System.out.println("Come rato");
    }

}
```

É um Felino

É um Animal

Interfaces



É um **tipo especial** de classe contendo **métodos abstratos** e **atributos finais**

Interfaces por natureza **são abstratas**

Define um meio público e padrão de **especificar** o **comportamento** das classes

Notação UML

«interface» **Calculos**
+ multiplicacao(Double, Double) : Double + soma(Double, Double) : Double + subtracao(Double, Double) : Double

Criando Interfaces

[public] [abstract] interface <NomeDaInterface> {

[public] [final] <tipoAtributo> <atributo> = <valorInicial>;

[public] [abstract] <retorno> <nomeMetodo>(<parametro>*);

[public] default <retorno> <nomeMetodo>(<parametro>*){...}

[public] static <retorno> <nomeMetodo>(<parametro>*){...}

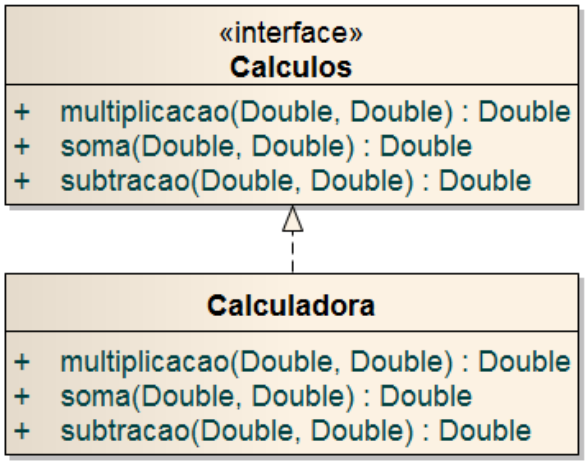
```
public interface Calculos {

    public Double soma(Number x, Number Y);

    public Double subtracao(Number x, Number Y);

    public Double multiplicacao(Number x, Number Y);
}
```

Implementando Interfaces

Palavra reservada **implements** e usada para implementar uma interface

```
public class Calculadora implements Calculos {

    @Override
    public Double soma(Double x, Double y) {
        return x + y;
    }

    @Override
    public Double subtracao(Double x, Double y) {
        return x - y;
    }

    @Override
    public Double multiplicacao(Double x, Double y) {
        return x * y;
    }
}
```

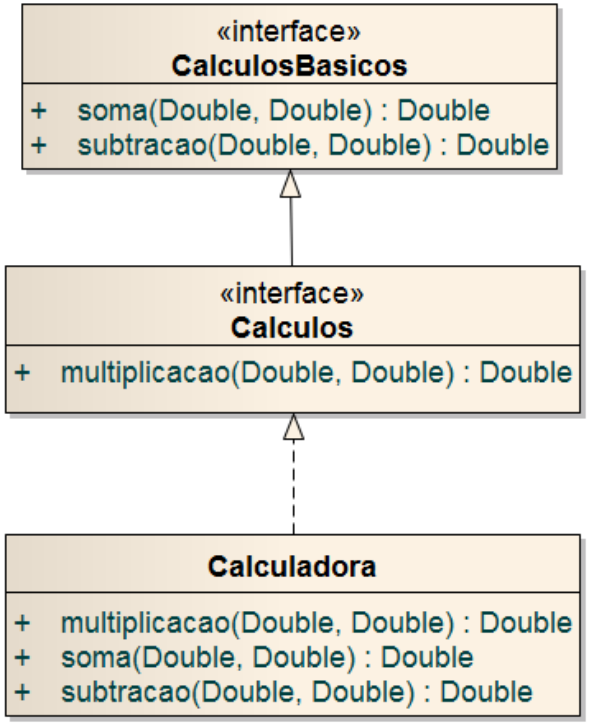
Notação UML



Herança entre interfaces

Interfaces não são partes da hierarquia de classe. Entretanto, interfaces podem ter relacionamentos de ***herança entre elas*** *próprias*

Notação UML




```
public interface CalculosBasicos {

    public Double soma(Double x, Double y);

    public Double subtracao(Double x, Double y);
}


public interface Calculos extends CalculosBasicos{

    public Double multiplicacao(Double x, Double y);

}
```

Interface vs. Classe

Interfaces e classes são tipos

Uma interface pode ser usada em lugares onde pode se usar uma classe

```
Calculos calcula = new Calculadora();

Calculadora calculadora = new Calculadora();

//Calculos calculos = new Calculos(); // Erro
```

*Não é permitido criar **instância** de uma **interface***

Classes Internas (Aninhadas)

Classe interna (**nested class**) é um recurso que permite definir uma classe dentro de outra.

```
public class ClasseExterna {

      public class ClasseInterna {
            public String toString() {
                  return "Classe Interna";
            }
      } // ClasseInterna

      public String toString() {
            ClasseInterna ci = new ClasseInterna();
            return "Classe Externa com " + ci;
      } // toString

      public static void main(String[] args) {
            ClasseExterna ce = new ClasseExterna();
            System.out.println(ce);
      } // main

} // ClasseExterna
```

Classes Interna Anônima

Classe interna sem nome com propósito de escopo limitado, utilizado para declarar e instanciar um objeto um única vez

```
public class AnonymousInnerClass {
    // Inner class
    class Fruta {
        public String nome;
    };

    public void ordenar(List<Fruta> frutas) {
        // Ordenar
        Collections.sort(frutas,
                // Anonymous Inner Class
                new Comparator<Fruta>() {
                    @Override
                    public int compare(Fruta fruta2, Fruta fruta1) {

                        return fruta1.nome.compareTo(fruta2.nome);
                    }
                });

    }
}
```

Objeto definido e criado no parâmetro

Java 8 - Lambda

•É um bloco de código com parâmetro:
(String p, String s) -> Integer.compare(p.length(),s.length())
podemos usar para simplificar digitação de classes anônimas na implementação de interfaces funcionais.

```java
// Ordenar
Collections.sort(frutas,
        // Anonymous Inner Class
        new Comparator<Fruta>() {
            @Override
            public int compare(Fruta fruta2, Fruta fruta1) {

                return fruta1.nome.compareTo(fruta2.nome);
            }
        });
```

Menos código

```java
@FunctionalInterface
public interface Comparator<T>
```

Interface Funcional, só tem um único método abstrato

```java
// Ordenar
Collections.sort(frutas,
        (fruta1, fruta2) ->
                fruta1.nome.compareTo(fruta2.nome)
        );
```

Lambda Métodos de Referência

Lambda com métodos que já existem, não são anônimos.
Você pode referenciar pelo nome pode ser

Classe::nomeMétodoEstático
instanciaClasse::nomeMétodoInstância
Classe::new (referência ao construtor)

```
// Ordenar
Collections.sort(frutas,
        (fruta1, fruta2) ->
                fruta1.nome.compareTo(fruta2.nome)
        );
```

Método anônimo na expressão lambda

```
public int compararPeloNome(Fruta f1, Fruta f2) {
    return f1.nome.compareTo(f2.nome);
}
```

Método de instância definido numa classe

```
Collections.sort(frutas,refClass::compararPeloNome);

    frutas.forEach(System.out::print);
```

Método de referência usado numa expressão lambda

Método de referência, estático

Tipos Enumerados

O tipo **enum** estende implicitamente a classe **java.lang.Enum.** As enumerações podem ter construtores, métodos, variáveis. Cada elemento é uma instância do enum.

```
public enum EnumEstacoes {

    PRIMAVERA(EnumMes.SETEMBRO, EnumMes.NOVEMBRO),
    VERAO(EnumMes.DEZEMBRO, EnumMes.FEVEREIRO),
    OUTONO(EnumMes.MARCO, EnumMes.MAIO),
    INVERNO(EnumMes.JUNHO, EnumMes.AGOSTO);

    private EnumMes inicio, fim;

    private EnumEstacoes( EnumMes inicio, EnumMes fim ) {
        this.inicio = inicio;
        this.fim = fim;
    }

    // somente métodos get são necessários
    public EnumMes getInicio() { return inicio; }
    public EnumMes getFim() { return fim; }

    enum EnumMes {
        JANEIRO, FEVEREIRO, MARCO, ABRIL, MAIO, JUNHO, JULHO,
        AGOSTO, SETEMBRO, OUTUBRO, NOVEMBRO, DEZEMBRO;
    }
}
```

- Como as classes, as enumerações podem ser declaradas dentro de classes, como estáticas e locais.