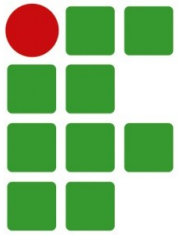


INSTITUTO FEDERAL
Goiás

Bacharelado em Sistemas de Informação
Disciplina: Programação Orientada a Objetos I

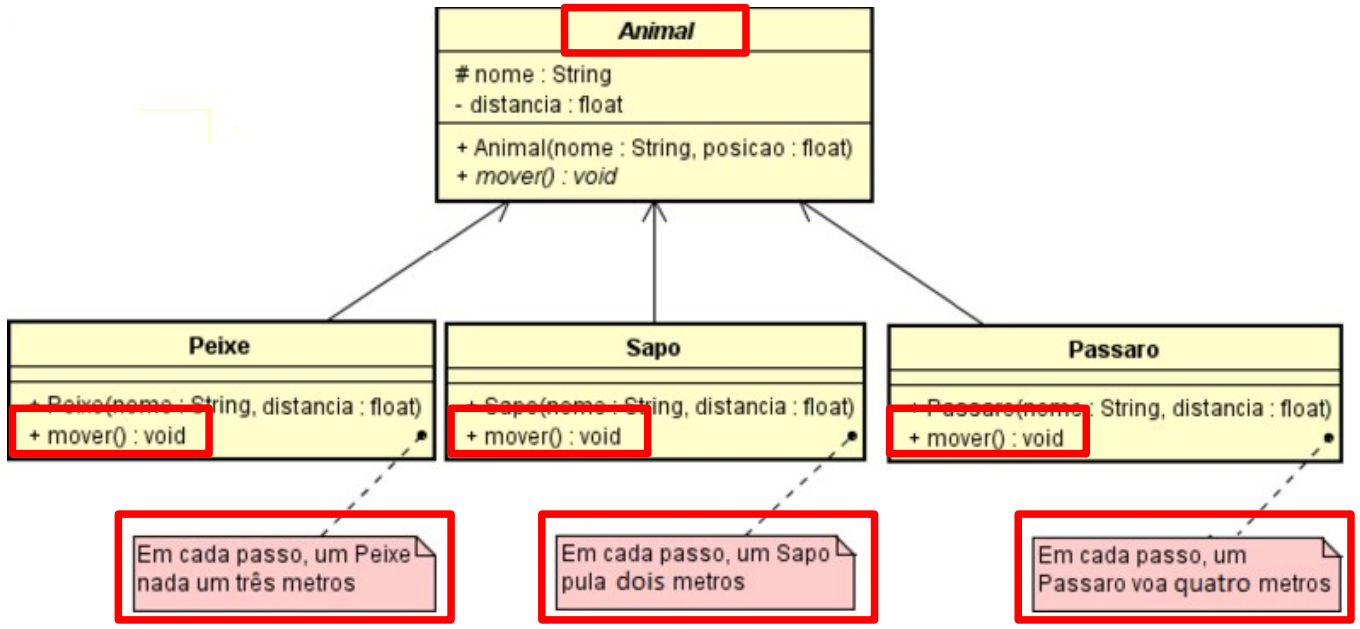
Associação de Classes
Herança
Polimorfismo

Prof. Ms. Dory Gonzaga Rodrigues
Goiânia - GO

Polimorfismo

Permite que uma mesma operação possa ser definida para diferentes tipos de classes, e cada uma delas a implementa como quiser.

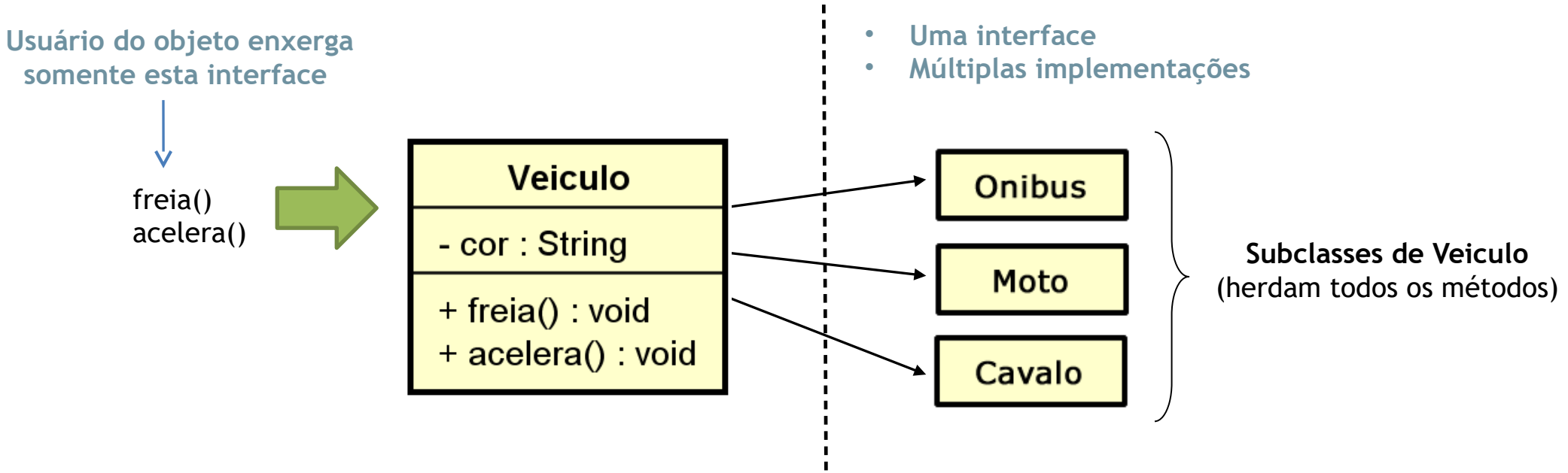
Princípio pelo qual duas ou mais classes derivadas de uma mesma superclasse podem invocar métodos que têm a mesma assinatura (lista de argumentos e tipo de retorno) mas com comportamentos distintos (especializados em cada subclasse).



- Permite a um mesmo objeto se manifestar de diferentes formas;
- Em linguagens fortemente tipadas o polimorfismo é implementado através de herança ou implementação de interfaces.
- O método será selecionado de acordo com o tipo da classe que instancia o objeto;

Polimorfismo

Um objeto que faz papel de interface serve de intermediário fixo entre o programa cliente e os objetos que, de fato, irão executar as mensagens recebidas.

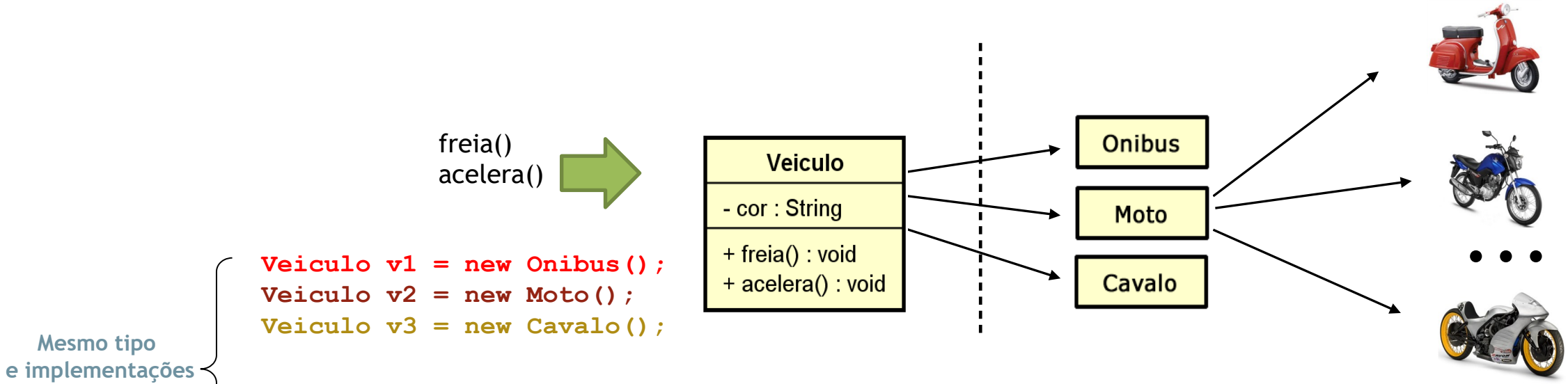


- O programa cliente não precisa saber da existência de outros objetos;
- Objetos podem ser substituídos sem que o programa cliente (que usa esta interface) seja afetado.

Polimorfismo

Novos objetos podem ser usados em programas que NÃO previam sua existência.

- Garantia que métodos da interface existem nas classes novas;
- Objetos de novas classes podem ser criados e usados (programa pode ser estendido durante a execução).

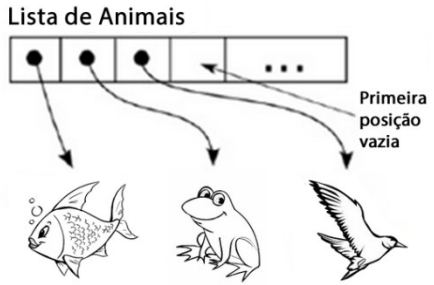


Mesmo tipo e implementações diferentes

```
Veiculo v1 = new Onibus();
Veiculo v2 = new Moto();
Veiculo v3 = new Cavalo();

v1.acelera(); //Acelera Onibus
v2.acelera(); //Acelera Moto
v3.acelera(); //Acelera Cavalo
```

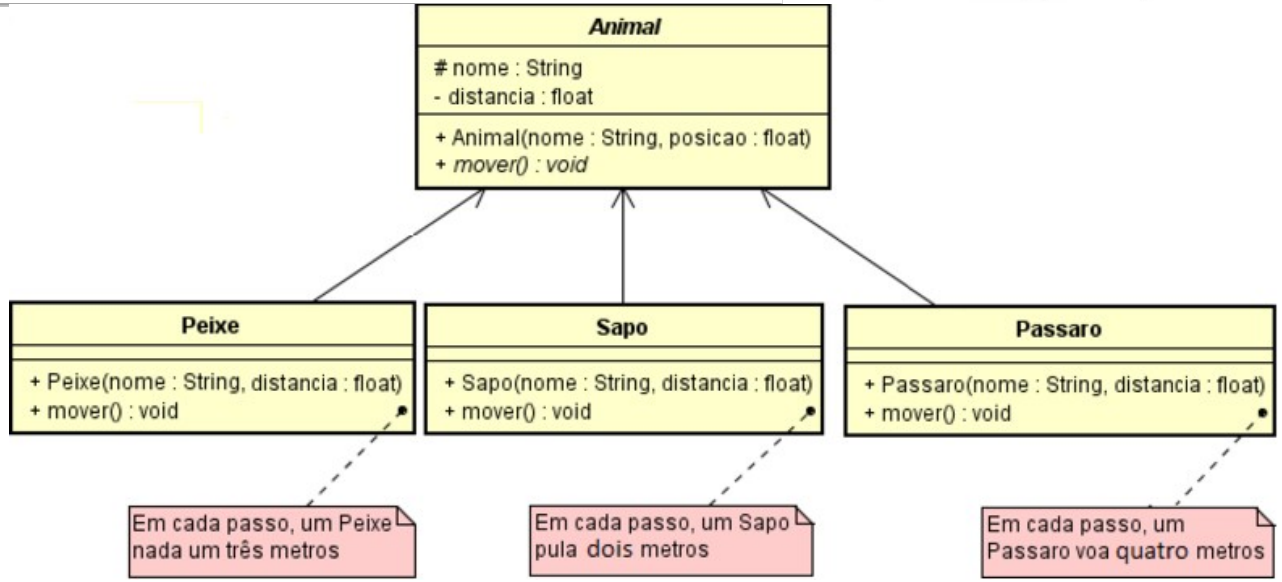
Exemplo

Permite que uma mesma operação possa ser definida para diferentes tipos de classes, e cada uma delas a implementa como quiser.



As classes Peixe, Sapo e Pássaro representam os três tipos de animais sob investigação.

- Cada classe estende a superclasse Animal, que contém um método mover;
- Toda subclasse implementa o método mover;

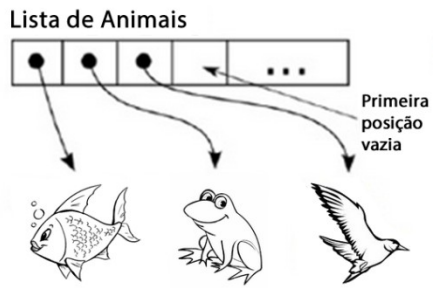


- Um programa mantém uma lista de Animal que contém referências a objetos das várias subclasses Animal. Para simular os movimentos dos animais, o programa envia a mesma mensagem, em cada passo, para cada objeto - a saber, move.

Exemplo

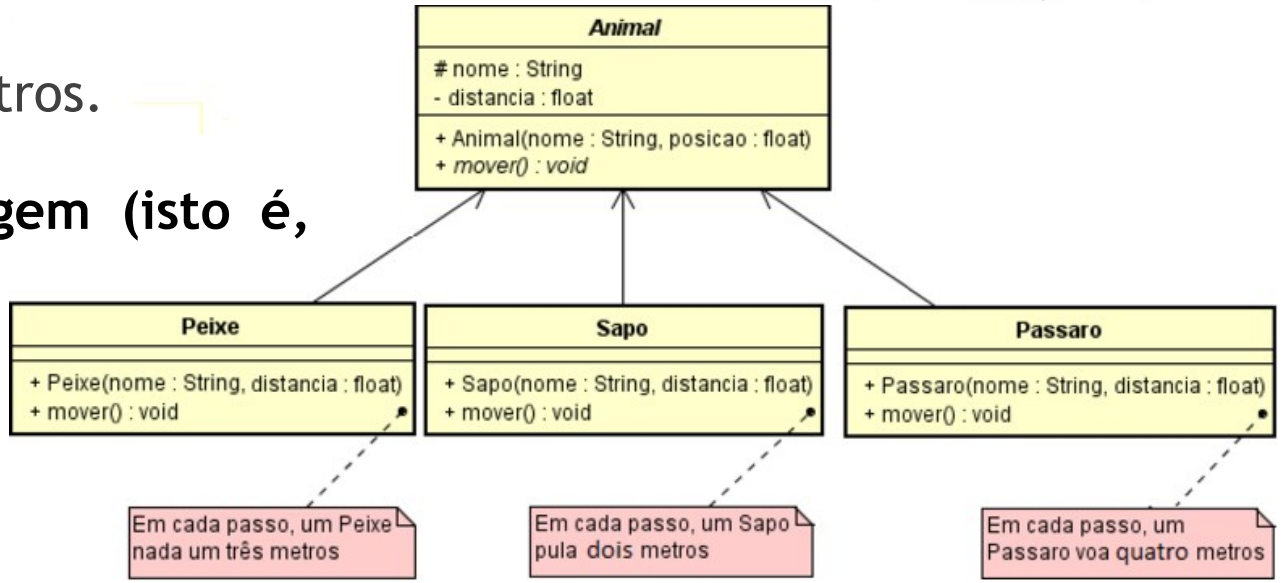
Cada tipo específico de Animal responde a uma mensagem mover de uma maneira única:

- um Peixe poderia nadar três metros;
- um Sapo poderia pular dois metros;
- um Pássaro poderia voar quatro metros.



O programa emite a mesma mensagem (isto é, mover) para cada objeto animal.

- Cada objeto sabe como modificar sua posição apropriadamente de acordo com seu tipo específico de movimento.

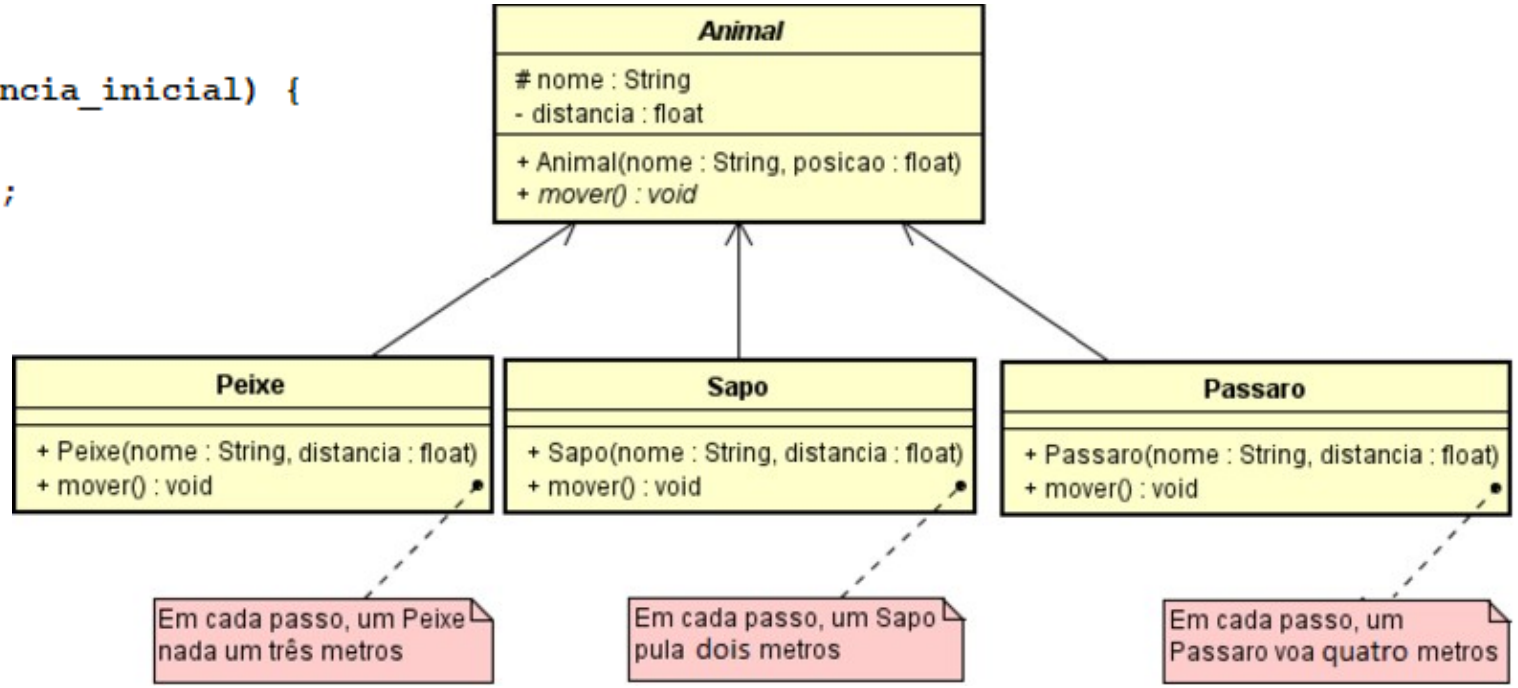


- Contar com o fato de que cada objeto sabe como "agir corretamente" em resposta à mesma chamada de método é o conceito-chave do polimorfismo.
- A mesma mensagem enviada a uma variedade de objetos tem "muitas formas" de resultados - daí o termo polimorfismo.

Polimorfismo

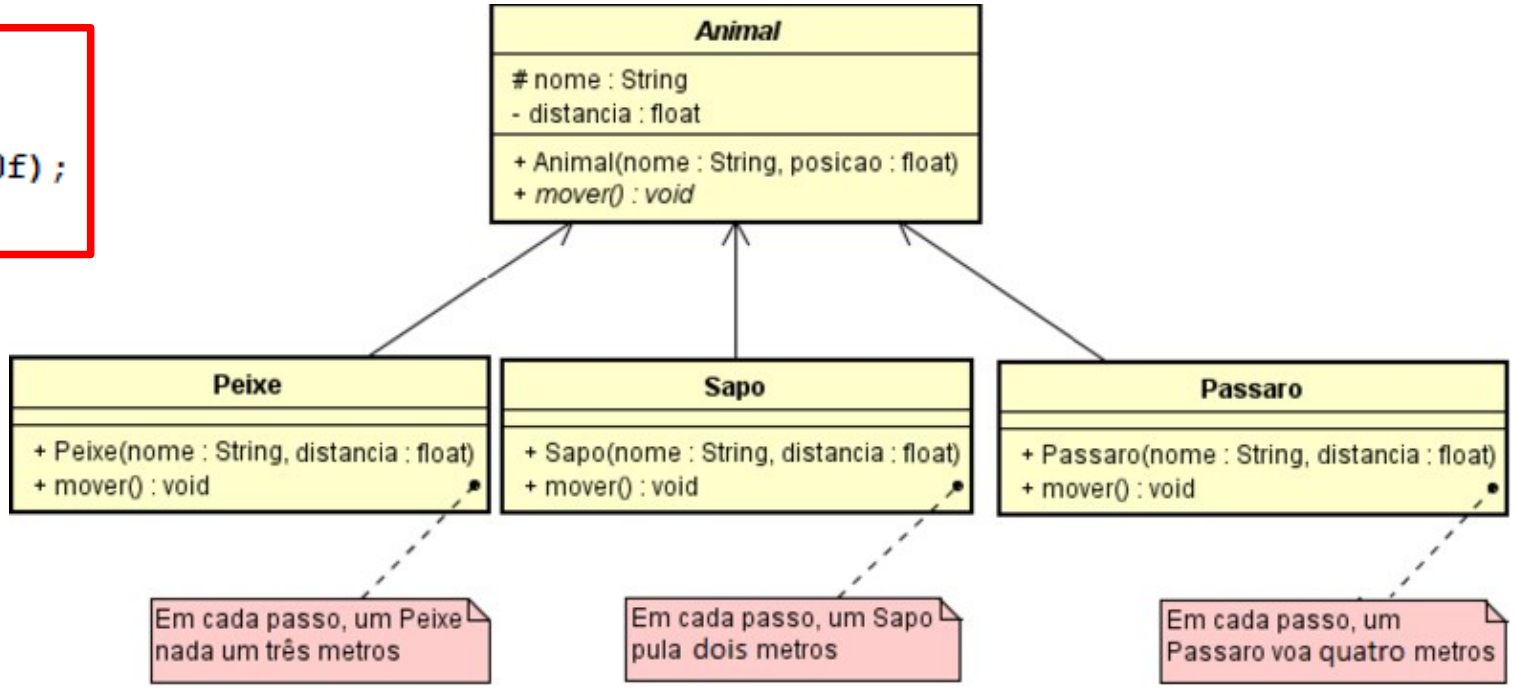
Implementando a Classe Animal

```
public abstract class Animal {
    protected String nome;
    private float distancia;

    public Animal(String nome, float distancia_inicial) {
        this.nome = nome;
        this.distancia = distancia_inicial;
    }

    abstract public void mover();

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public float getDistancia() {
        return distancia;
    }

    public void setDistancia(float distancia_inicial) {
        this.distancia = distancia_inicial;
    }
}
```



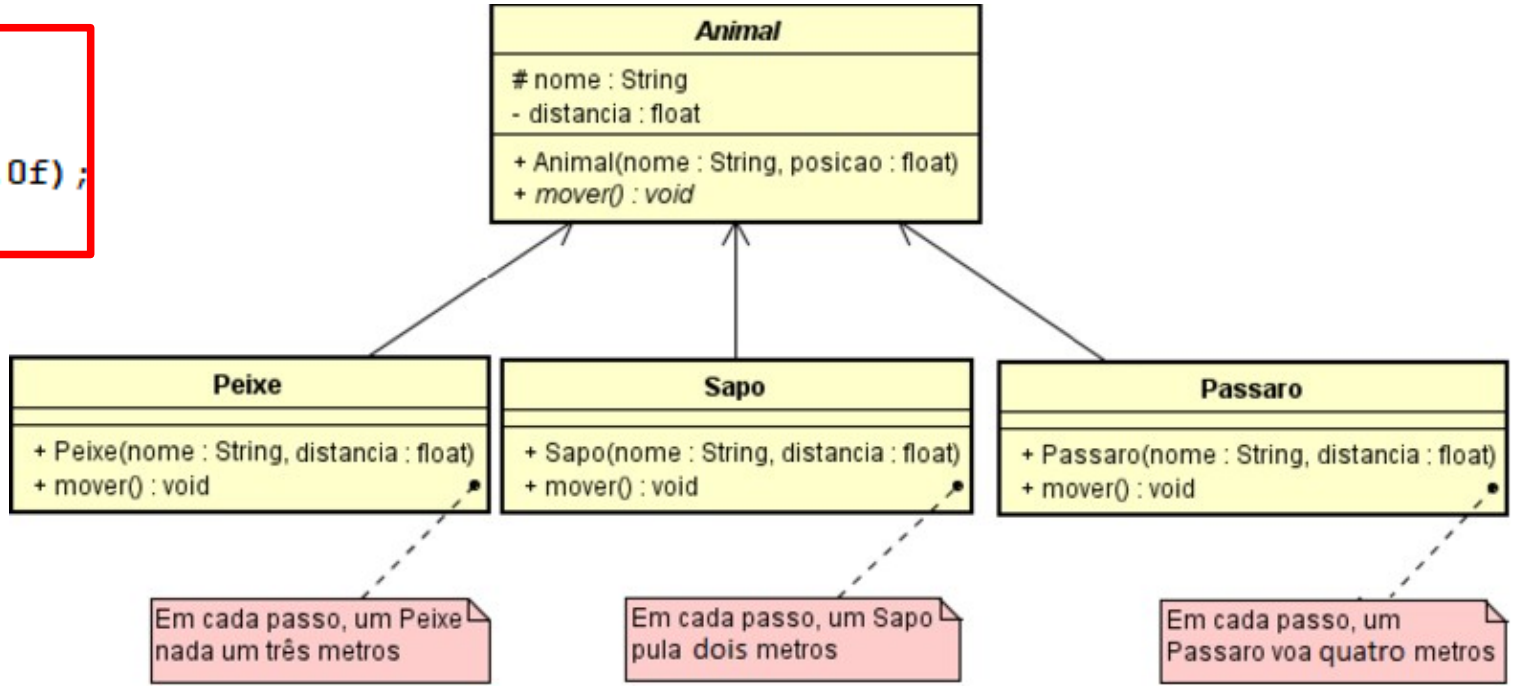
Polimorfismo
Implementando a Classe Peixe

```
public class Peixe extends Animal {

    public Peixe(String nome, float distancia_inicial) {
        super(nome, distancia_inicial);
    }

    @Override
    public void mover() {
        setDistancia(getDistancia() + 3.0f);
    }
}
```



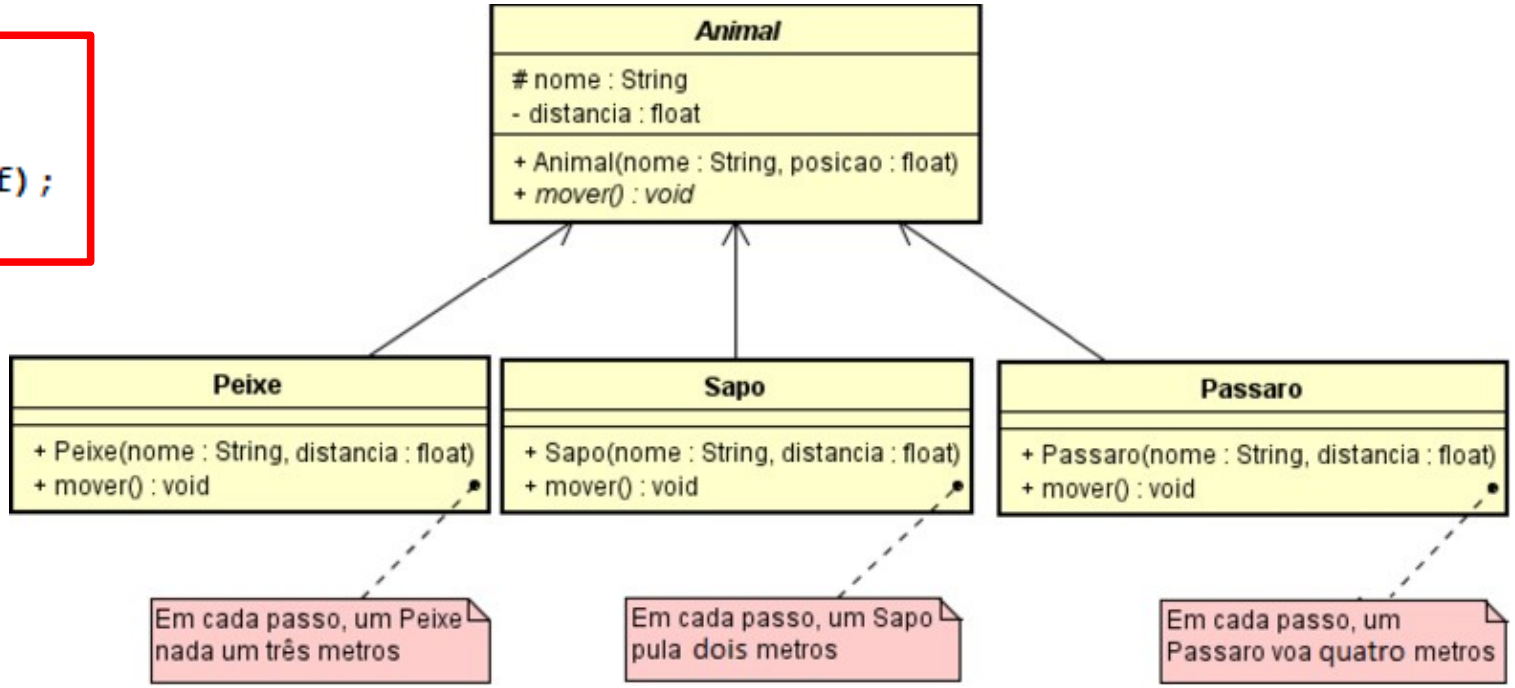
Polimorfismo
Implementando a Classe Sapo

```
public class Sapo extends Animal {

    public Sapo(String nome, float distancia_inicial) {
        super(nome, distancia_inicial);
    }

    @Override
    public void mover() {
        setDistancia(getDistancia() + 2.0f);
    }
}
```



Polimorfismo
Implementando a Classe Pássaro

```
public class Passaro extends Animal {

    public Passaro(String nome, float distancia_inicial) {
        super(nome, distancia_inicial);
    }

    @Override
    public void mover() {
        setDistancia(getDistancia() + 4f);
    }
}
```



Polimorfismo

TesteAnimal (Programa 1)

```
Console
<terminated> TesteAnimal [Java Application] C:\Program Files\Java\jdk-11.0.4\bir
--- Animais e a distância de seus passos -----
Tucunaré: 3.0 metros !
Cururu: 2.0 metros !
Andorinha: 4.0 metros !
```

```java
import java.util.ArrayList;
import java.util.List;

public class TesteAnimal {
    public static void main(String[] args) {
    List<Animal> listaAnimais = new ArrayList<Animal>();

    Animal peixe = new Peixe("Tucunaré", 0);
    Animal sapo = new Sapo("Sapo-Comum", 0);
    Animal passaro = new Passaro("Andorinha", 0);

    listaAnimais.add(peixe);
    listaAnimais.add(sapo);
    listaAnimais.add(passaro);
    System.out.println("---- Animais e as distâncias de seus Passos ----");

    for (Animal animal : listaAnimais) {
            animal.mover();
            System.out.println(animal.getNome() + ": " + animal.getDistancia() + " metros !");
        }
    }
}
```

Polimorfismo

TesteAnimal2 (Programa 2)

```
1  import java.util.ArrayList;
3
4  public class TesteAnimal2 {
5      public static void main(String[] args) {
6
7          List<Animal>  listaAnimais = new ArrayList<>();
8          Animal  a1 = new Peixe("Tucunaré", 0.0f);
9          Animal  a2 = new Sapo("Cururu", 0.0f);
10         Animal  a3 = new Passaro("Andorinha", 0.0f);
11
12         listaAnimais.add(a1);
13         listaAnimais.add(a2);
14         listaAnimais.add(a3);
15         a1.mover();
16         a2.mover();
17         a3.mover();
18
19         System.out.println("--- Lista de Animais -----");
20         for (Animal animal : listaAnimais) {
21             if (animal.getClass().getName() == "Passaro")
22                 System.out.println("Pássaro : " +  animal.getNome() + "   " + animal.getDistancia() );
23             if (animal.getClass().getName() == "Sapo")
24                 System.out.println("Sapo : " + animal.getNome() + "   " + animal.getDistancia() );
25             if (animal.getClass().getName() == "Peixe")
26                 System.out.println( "Peixe : " + animal.getNome() + "   " + animal.getDistancia() );
27         }
28     }
29 }
```

Console

<terminated> TesteAnimal2 [Java Application]

```
--- Lista de Animais -----
Peixe : Tucunaré   3.0
Sapo : Cururu   2.0
Pássaro : Andorinha   4.0
```

Polimorfismo

O polimorfismo permite escrever programas que processam objetos que compartilham a mesma superclasse como se todas fossem objetos da superclasse.

- O polimorfismo simplifica a programação, permite projetar e implementar sistemas que são facilmente extensíveis.
- Novas classes podem ser adicionadas com pouca ou nenhuma modificação a partes gerais do programa, contanto que as novas classes sejam parte da hierarquia de herança que o programa processa genericamente.
- As únicas partes de um programa que devem ser alteradas para acomodar as novas classes são aquelas que exigem conhecimento direto das novas classes que adicionamos à hierarquia.