Programação Orientada a Objetos

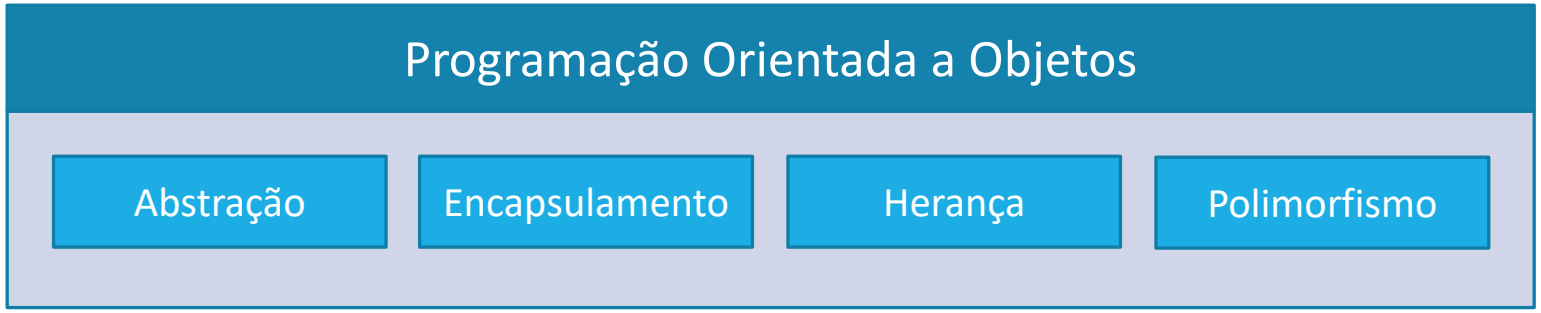
PROFESSOR ISAAC

Polimorfismo

Polimorfismo

É um dos pilares da Orientação a Objetos.

Pilares da Orientação a Objetos



Polimorfismo: Traduzindo do grego, significa "*muitas formas*".

Fonte: http://www.devmedia.com.br/os-4-pilares-da-programacao-orientada-a-objetos/9264

Conceitos de Polimorfismo

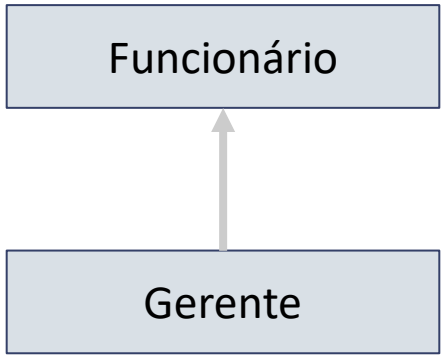
- **Polimorfismo** é a capacidade de um **objeto** poder ser **referenciado de várias formas**.

- O **polimorfismo** está ligado aos conceitos de **Herança** e **Hierarquia de Classes**

Conceitos de Polimorfismo

- Exemplo:

Podemos nos referir a um Gerente como sendo um Funcionário:




```
Funcionario func;
func = new Gerente();
```

- O objeto *func* é declarado como um Funcionário, mas é instanciado como um Gerente!
- *func* vai agir como um Gerente!

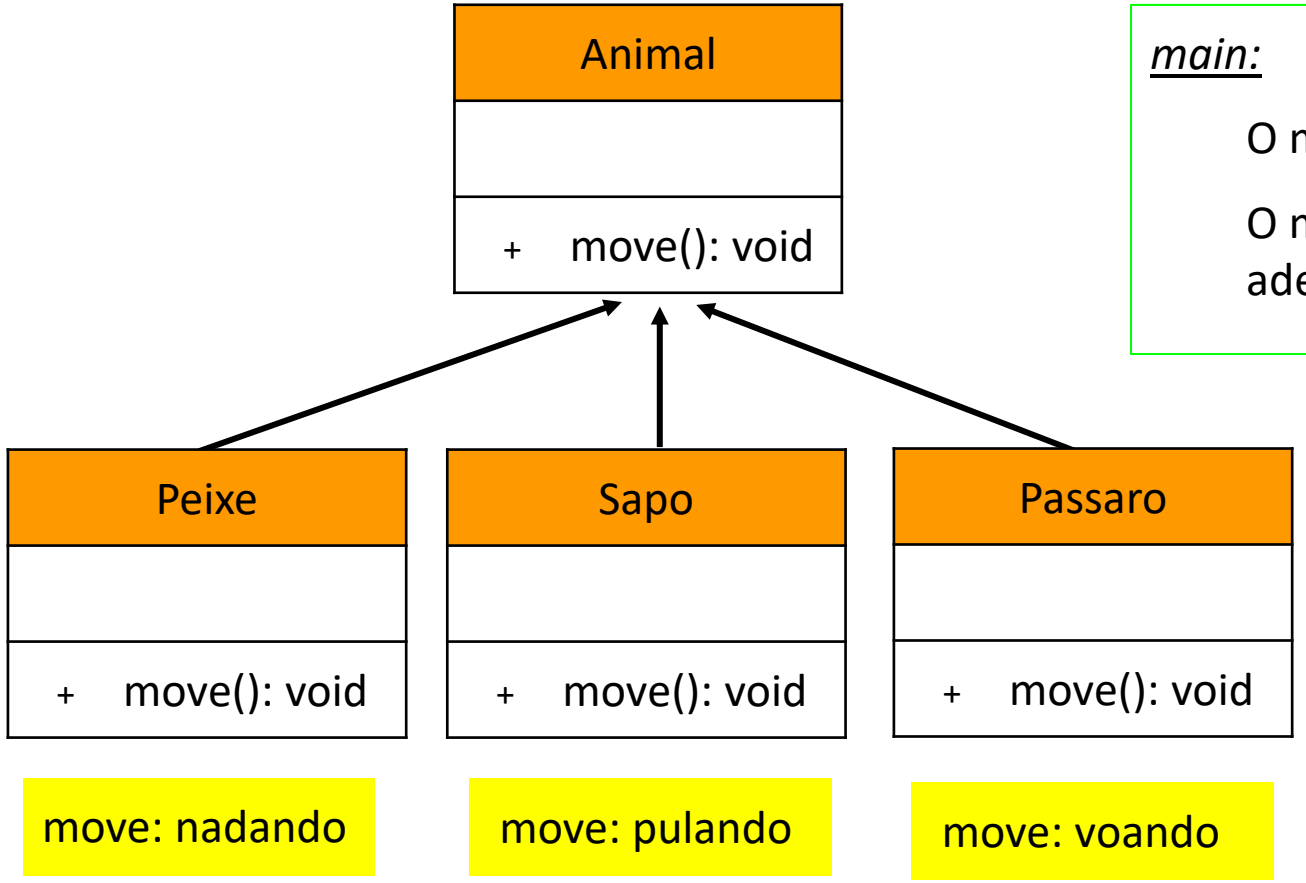
Conceitos de Polimorfismo

- Permite **programar no geral** em vez de **programar no específico.**
- Processa objetos de classes que fazem parte da mesma hierarquia, como se fossem todos objetos da superclasse.

Método polimórfico

- **Cada objeto** sabe fazer a coisa certa em **resposta à mesma chamada de método:**
  - Diferentes ações ocorrem, dependendo do tipo de objeto!
  - Os métodos são polimórficos!

Exemplo - Hierarquia Animal



main:

O método *move()* é polimórfico!

O método responde de forma adequada para cada animal.

Exemplo - Hierarquia Animal

```java
class Animal {
    public void move() {
        System.out.println( "Animal anda" );
    }
}

class Peixe extends Animal {
    @Override
    public void move() {
        System.out.println( "Nada" );
    }
}

class Sapo extends Animal {
    @Override
    public void move() {
        System.out.println( "Pula" );
    }
}

class Passaro extends Animal {
    @Override
    public void move() {
        System.out.println( "Voa" );
    }
}
```

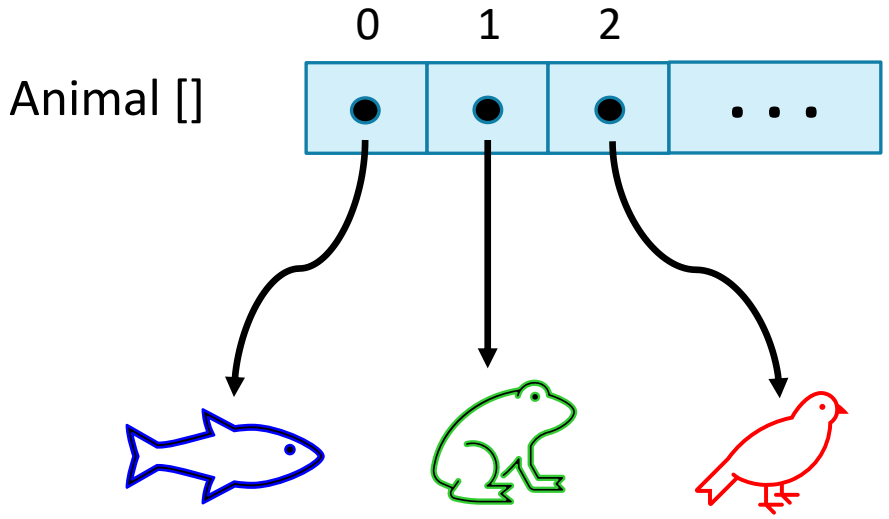
```java
import java.util.ArrayList;

public class MainAnimalPoli {
    public static void main( String[] args ) {
        ArrayList<Animal> animais = new ArrayList<Animal>();
        animais.add( new Peixe() );
        animais.add( new Sapo() );
        animais.add( new Passaro() );

        animais.get(0).move();
        animais.get(1).move();
        animais.get(2).move();
    }
}
```

@Override é uma anotação - serve para indicar que o método da subclasse está sendo sobrescrito / reescrito.

Exemplo - Hierarquia Animal




```java
import java.util.ArrayList;

public class MainAnimalPoli {
    public static void main( String[] args ) {
        ArrayList<Animal> animais = new ArrayList<Animal>();
        animais.add( new Peixe() );
        animais.add( new Sapo() );
        animais.add( new Passaro() );

        animais.get(0).move();
        animais.get(1).move();
        animais.get(2).move();
    }
}
```

Vantagens do Polimorfismo

- Escalabilidade:
  - Com o polimorfismo podemos projetar e implementar sistemas que são facilmente extensíveis.
  - Novas classes podem ser adicionadas com pouco ou nenhuma alteração no programa
- Clareza e manutenção do código com menos linhas e flexibilidade.

Classes Abstratas

Classes Abstratas

- A classe abstrata é sempre uma **superclasse** que não possui instâncias: **não pode ser instanciada.**
- **Ela define um modelo genérico** para determinada funcionalidade e geralmente fornece uma implementação incompleta dessa funcionalidade.
- Cada uma das subclasses da classe abstrata completa a funcionalidade da classe abstrata, adicionando um comportamento específico.

Classes Abstratas

- No Java precisamos especificar explicitamente que a classe é abstrata, através da palavra-chave **abstract**.

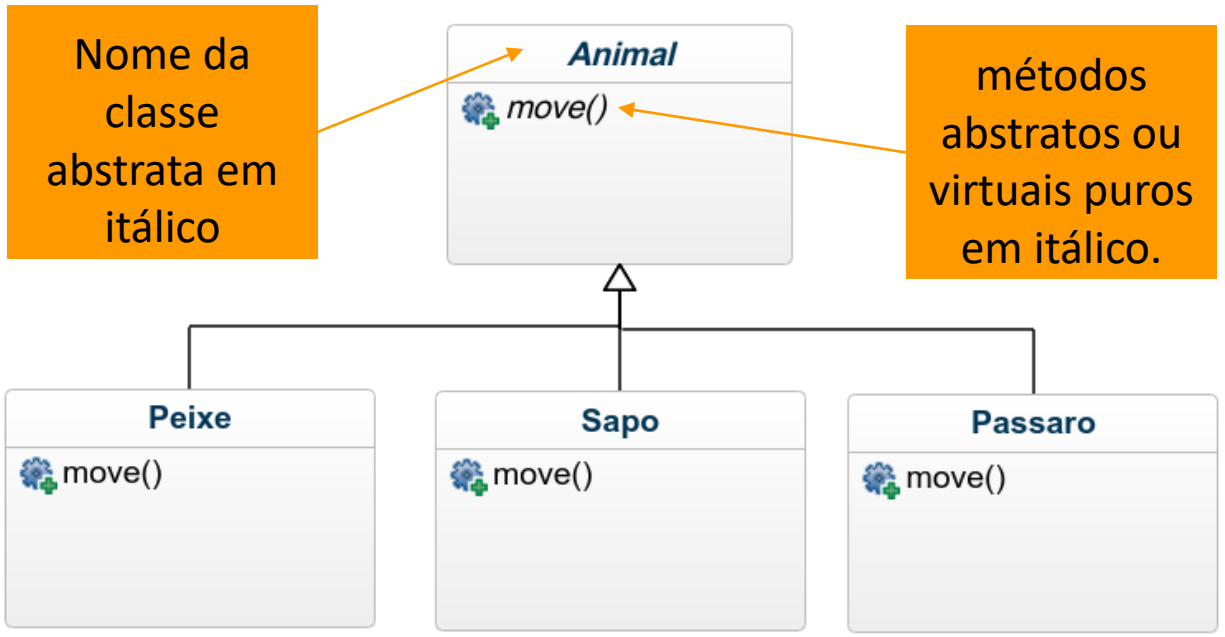
- Classes abstratas normalmente contém um ou mais métodos abstratos, que também são especificados por **abstract.**
  - Métodos abstratos não têm implementação!
  - Definir um método como abstrato é uma maneira de forçar a sua implementação nas subclasses

Classes Abstratas

- No nosso exemplo, a classe **Animal** ficaria:

```
1  abstract class Animal {
2       public abstract void move();
3  }
4
5  class Peixe extends Animal {
6       @Override
7       public void move() {
8           System.out.println( "Nada" );
9       }
10 }
11
12 class Sapo extends Animal {
13      @Override
14      public void move() {
15          System.out.println( "Pula" );
16      }
17 }
18
19 class Passaro extends Animal {
20      @Override
21      public void move() {
22          System.out.println( "Voa" );
23      }
24 }
```

UML - Diagrama de Classes
Classe Abstrata



Exemplo completo: Polimorfismo, Classe Abstrata

```java
abstract class OperacaoMatematica {
    public abstract double calcular(double x, double y);
}

class Soma extends OperacaoMatematica {
    public double calcular(double x, double y) {
        return x + y;
    }
}

class Subtracao extends OperacaoMatematica {
    public double calcular(double x, double y) {
        return x - y;
    }
}
```

```java
import java.util.Scanner;

class Contas {

    public static void main(String args[]) {
        OperacaoMatematica operacao;

        Scanner entrada = new Scanner( System.in );
        String op = entrada.nextLine();

        if (op.equals("soma"))
            operacao = new Soma();
        else
            operacao = new Subtracao();

        System.out.println("Resultado: " + operacao.calcular(5, 4));
    }
}
```

Exercício

- Implemente as classes multiplicar e dividir com o método calcular.

Upcasting / Downcasting

Casting

- Na linguagem Java, é comum termos que atribuir o valor de um tipo de variável a outro tipo de variável, porém para tal é necessário que esta operação seja apontada ao compilador. A este apontamento damos o nome de casting.

Exemplo:

```
//Conversão de double para int

int p = (int) 3.14;
```

Casting

- Temos essa conversão que pode ocorrer de forma implícita ou explícita.

Exemplo de forma explícita:

```
//Conversão de double para int

int p = (int) 3.14;
```

Exemplo de forma implícita:

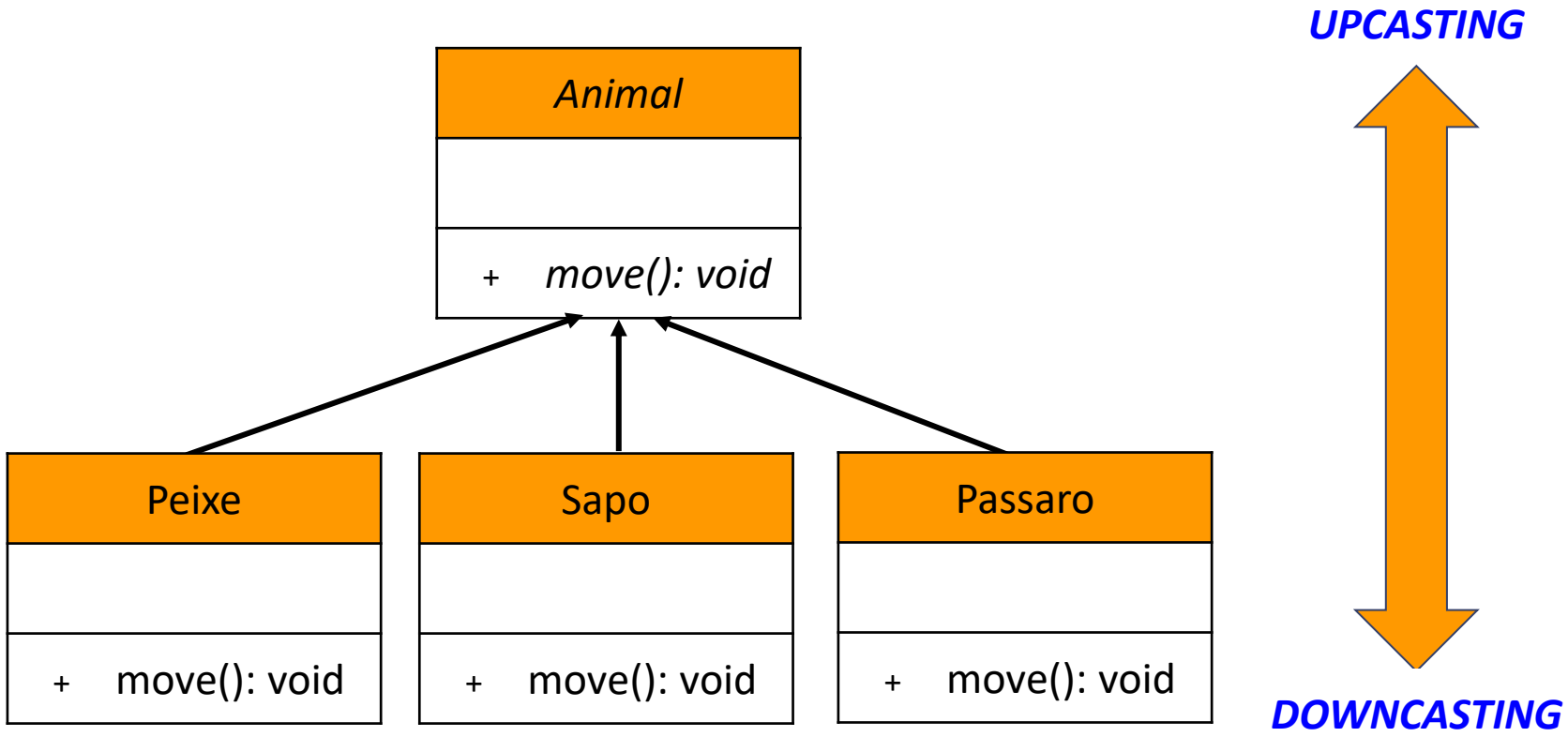
```
//Conversão de char para int

int p = 'a' ;
```

Upcasting / Downcasting

- Podem ser feitos por meio do **Polimorfismo**
- Tem relação direta com **Herança** e **Hierarquia de Classes**

Exemplo - Hierarquia Animal



Upcasting

Com o upcasting, podemos nos referir a um objeto da subclasse como sendo da superclasse

Upcasting não precisa ser feito de forma **explícita**!

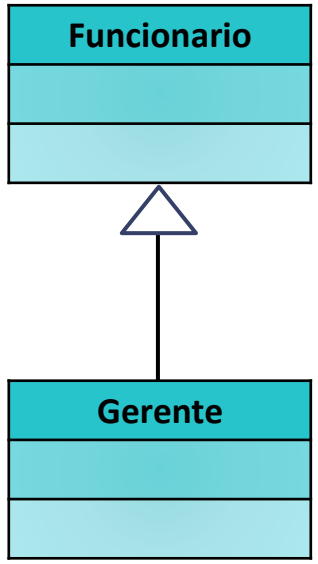
Upcasting

Já usamos:

Classe Base / Superclasse: Funcionario

Classe Derivada / Subclasse: Gerente

```
Funcionario g = new Gerente()
```



Upcasting

É como uma **promoção** do tipo Gerente para o tipo Funcionário

O upcasting de forma explícita seria:

```
Funcionario g = (Funcionario) new Gerente()
```

Downcasting

É o processo **contrário ao upcasting.**

A ideia aqui é converter a referência da superclasse para uma referência de uma subclasse.

Sempre precisa ser explícito!

Downcasting

Classe Base / Superclasse: Funcionario

Classe Derivada / Subclasse: Gerente

```
Funcionario g = new Gerente()
Gerente g2 = (Gerente) g
```

Por que usar Downcasting?

Exemplo - Hierarquia Animal

```java
class Animal {
    public void move() {
        System.out.println( "Animal anda" );
    }
}
```

```java
import java.util.ArrayList;

class Peixe extends Animal {
    public void move() {
        System.out.println( "Nada" );
    }

    public void fala(){
        System.out.println( "Peixe fala" );
    }
}

class Sapo extends Animal {
    public void move() {
        System.out.println( "Pula" );
    }
}

class Passaro extends Animal {
    public void move() {
        System.out.println( "Voa" );
    }
}
```

```java
class MainAnimalPoli {
    public static void main( String[] args ) {
        ArrayList<Animal> animais = new ArrayList<Animal>();

        animais.add( new Peixe() );
        animais.add( new Sapo() );
        animais.add( new Passaro() );

        animais.get(0).move();
        animais.get(1).move();
        animais.get(2).move();
        animais.get(0).fala();
    }
}
```

não funciona!

```
MainAnimalPoli.java:37: error: cannot find symbol
        animais.get(0).fala();
                       ^
  symbol:   method fala()
  location: class Animal
1 error
```

Exemplo - Hierarquia Animal


```
class Animal {
    public void move() {
        System.out.println( "Animal anda" );
    }
}
```

```
1  import java.util.ArrayList;
2
3  class Peixe extends Animal {
4      public void move() {
5          System.out.println( "Nada" );
6      }
7
8      public void fala(){
9          System.out.println( "Peixe fala" );
10     }
11 }
12
13 class Sapo extends Animal {
14     public void move() {
15         System.out.println( "Pula" );
16     }
17 }
18
19 class Passaro extends Animal {
20     public void move() {
21         System.out.println( "Voa" );
22     }
```

```
class MainAnimalPoli {
    public static void main( String[] args ) {
        ArrayList<Animal> animais = new ArrayList<Animal>();

        animais.add( new Peixe() );
        animais.add( new Sapo() );
        animais.add( new Passaro() );

        animais.get(0).move();
        animais.get(1).move();
        animais.get(2).move();
        ((Peixe)animais.get(0)).fala();
    }
}
```

Funciona!!!

```
Nada
Pula
Voa
Peixe fala
```

Exemplo - Hierarquia Animal


```
ArrayList<Animal> animais = new ArrayList<>();

animais.add(new Peixe());
animais.add(new Passaro());
animais.add(new Sapo());
animais.add(new Peixe());

for (Animal obj : animais) {
    if (obj instanceof Peixe) {
        ((Peixe) obj).mergulha();
    } else if (obj instanceof Sapo) {
        ((Sapo) obj).come();
    } else if (obj instanceof Passaro) {
        ((Passaro) obj).fala();
    } else {
        System.out.println("Don't know what to do with a " + obj.getClass());
    }
}
```