

# labWeek6

Aluno: Arthur Calciolari

Grupo: MasterClass A

Professora: Tatiana Pereira

---

Polimorfismo:

▼ Sobrecarga:

▼ O conceito de sobrecarga consiste em criar vários métodos com o mesmo nome, porém com parametros diferentes. O método desejado será chamado com base na quantia de parâmetros:

```
//polimorfismo de sobrecarga;  
package Polimorfismo;  
  
no usages  
public class polSobrecarga {  
    no usages  
    public void calculo(){  
        // liberado  
    }  
    no usages  
    public void calculo(int x){  
        // mesmo nome, porem o parametro diferencia  
    }  
    no usages  
    public void calculo(int x, int y){  
        //mesmo nome das duas anteriores, porem os parametros os diferenciam;  
    }  
}
```

### ▼ Override:

O conceito de override consiste em reescrever um método. É necessário que ele tenha o mesmo retorno, nome e tipo de variável:

```
package Polimorfismo.TESTE;

no usages
public class Main {
    2 usages
    public class Veiculo {
        1 usage
        public int horsepower = 150;
        1 usage
        public void potencia(){
            System.out.println(horsepower);
        }
    }
    no usages
    public void main(String[] args) {
        Veiculo myCar = new Veiculo();
        // myCar.horsepower = 200;
        myCar.potencia();
    }
}
```

O output é 150;

```

package Polimorfismo.TESTE;

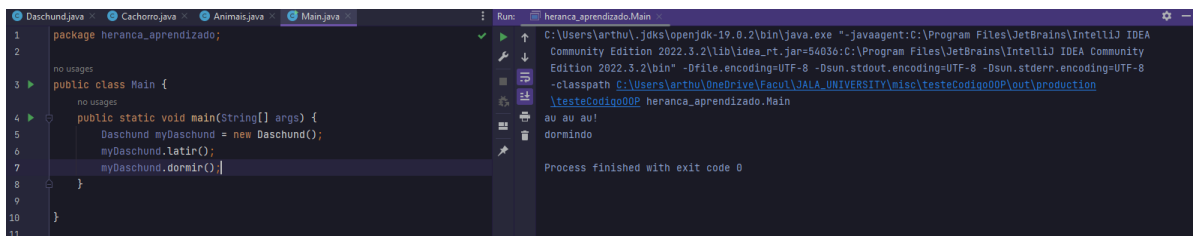
no usages
public class Main {
    2 usages
    public static class Veiculo {
        2 usages
        public int horsepower = 150;
        1 usage
        public void potencia(){
            System.out.println(horsepower);
        }
    }
    no usages
    public static void main(String[] args) {
        Veiculo myCar = new Veiculo();
        myCar.horsepower = 200;
        myCar.potencia();
    }
}

```

O output é 200 dessa vez, já que mudamos o valor da variável dentro da função Main.

### ▼ Universal de inclusão:

Método onde um mesmo objeto pode pertencer a diversas class simultaneamente, gerando uma herança hierárquica.



Ao instanciar o objeto Daschund, eu obtive acesso às funções que não estavam presentes nem nos métodos da classe Daschund e nem nos métodos da classe Cachorro, mas tive acesso aos métodos da classe Animais.

```
package heranca_aprendizado;

2 usages
public class Daschund extends Cachorro{
```

Classe Daschund sem nenhum método, porém foram herdados métodos passados

```
package heranca_aprendizado;

1 usage 1 inheritor
public class Cachorro extends Animais{ // o extends serve pra herdar os a

2 usages
    private boolean pedigree, domestico;

1 usage
    public void latir() { System.out.println("au au au!"); }

no usages
```

Classe cachorro que possui o método latir, mas não o dormir()

```
package heranca_aprendizado;

2 usages 3 inheritors
public class Animais {

2 usages
    private String Cor, raca, tamanho;

no usages
    public void comer() { System.out.println("comendo"); }

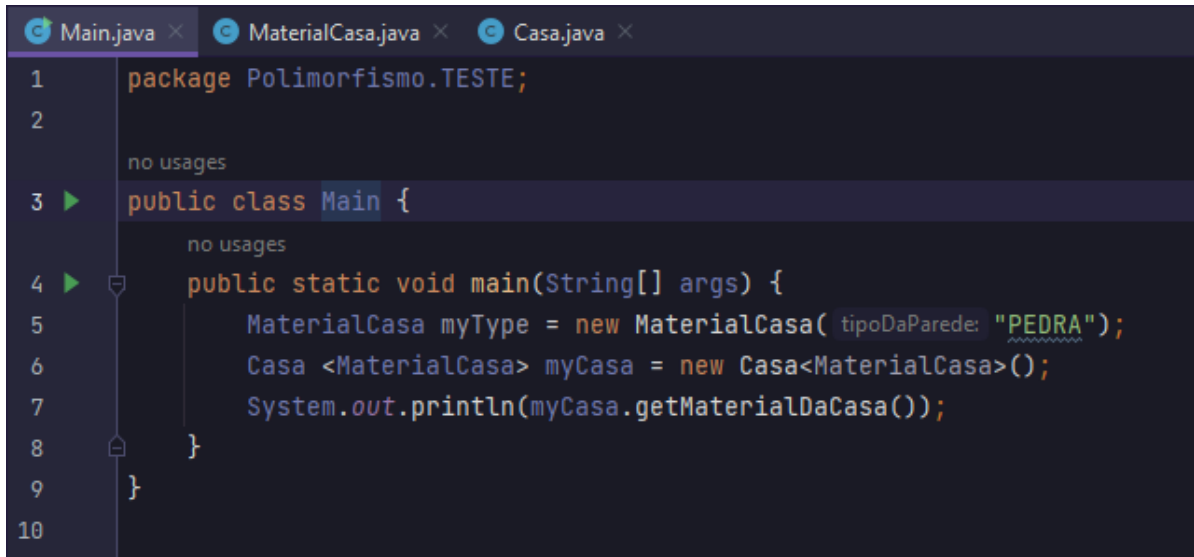
1 usage
    public void dormir() { System.out.println("dormindo"); }

no usages
```

Classe animais que possui o método dormir(), mas não o latir()

#### ▼ Universal paramétrico:

Serve para montar a estrutura genérica de um método, algo que você definirá depois no código. Um exemplo é quando você define uma casa, mas não especifica se ela será de madeira ou concreto.



```
1 package Polimorfismo.TESTE;
2
3 public class Main {
4     public static void main(String[] args) {
5         MaterialCasa myType = new MaterialCasa( tipoDaParede: "PEDRA");
6         Casa <MaterialCasa> myCasa = new Casa<MaterialCasa>();
7         System.out.println(myCasa.getMaterialDaCasa());
8     }
9 }
10
```

Fazendo isso, é possível deixar a estrutura montada e pedir um input ao usuário por exemplo. O polimorfismo universal paramétrico é uma técnica avançada que acaba por combinar um pouco de todos os outros polimorfismos.

#### ▼ Herança:

A herança serve para evitar a repetição de código. Suponhamos que temos uma classe Veículo, uma classe Audi e uma classe Volkswagen.

Para evitar a repetição de código nas classes Audi e Volkswagen, podemos utilizar o extends (em java) para herdar atributos e métodos de uma classe.

```
1 package Polimorfismo.TESTE;
2
3 public class Veiculo {
4
5     1 usage
6     public void corAudi(){
7         System.out.println("A audi é vermelha!");
8     }
9     no usages
10    public void corVolkswagen(){
11        System.out.println("O volkswagen é azul!");
12    }
```

Descrição da classe Veiculo;

```
1 package Polimorfismo.TESTE;
2
3 public class Audi extends Veiculo{
4 }
5
```

Descrição da classe Audi, mesmo ela estando vazia, ela herdou o métodos da classe veiculo;

```

1 package Polimorfismo.TESTE;
2
3 no usages
4 public class Main {
5     no usages
6     public static void main(String[] args) {
7         Audi myAudi = new Audi();
8         myAudi.corAudi();
9     }
10 }

```

Quando criamos o objeto, temos acesso aos métodos herdados dessa classe. O output nesse caso seria "A audi é vermelha!"