

# Herança e Polimorfismo

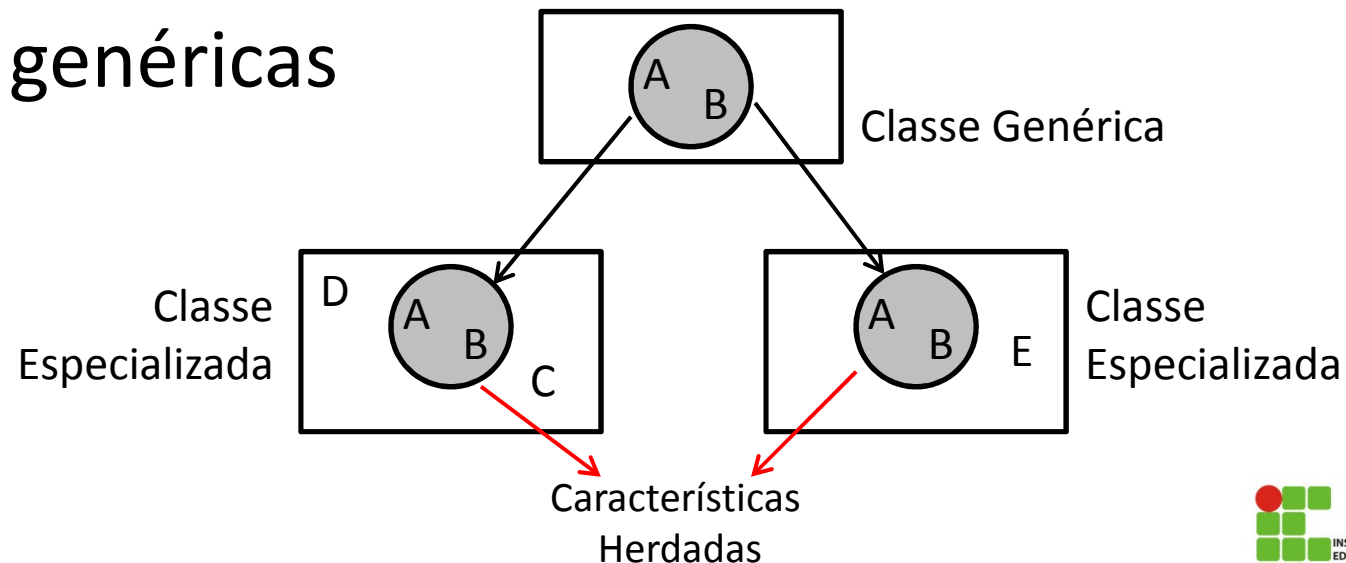


Prof. Bruno Gomes  
bruno.gomes@ifrn.edu.br

**Programação Orientada a Objetos**

# Revisando - Herança

- Estrutura Hierárquica e modular
- Projeção de classes genéricas que podem ser especializadas em classes mais particulares
- Classes especializadas reutilizam o código das mais genéricas



# Revisando - Herança

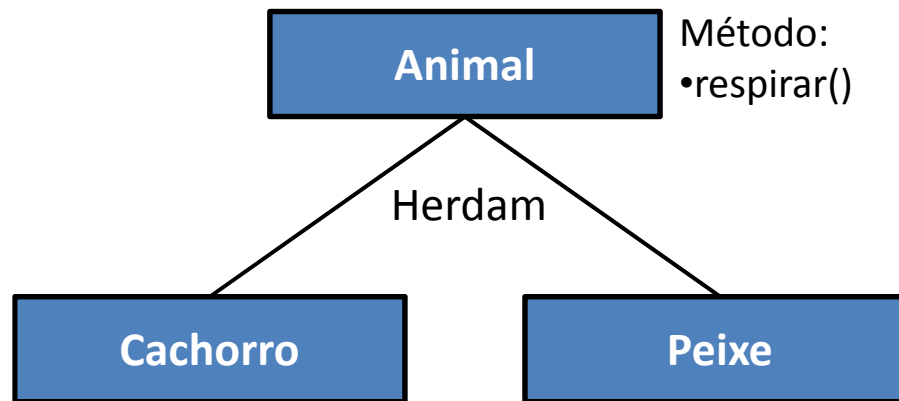
- Classe genérica, classe base, **superclasse** ou pai:
  - Define variáveis de instância “genéricas” e métodos
- Classe especializada, derivada, **subclasse** ou filha:
  - Especializa, **estende** ou herda os métodos “genéricos” de uma superclasse
  - Define apenas os métodos que são especializados

# Revisando - Polimorfismo

- Significa “várias formas”
- Habilidade de um mesmo tipo de objeto poder realizar ações diferentes ao receber uma mesma mensagem
- Criação de múltiplas classes com os mesmos métodos e propriedades, mas com funcionalidades e implementações diferentes
- Reescrita de código

# Revisando - Polimorfismo

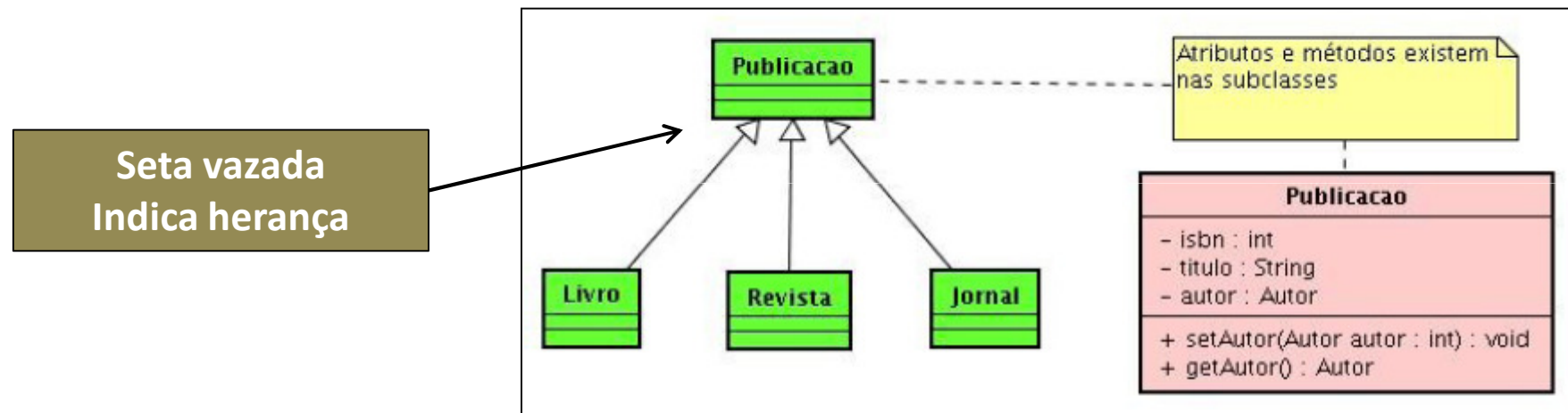
- Representação



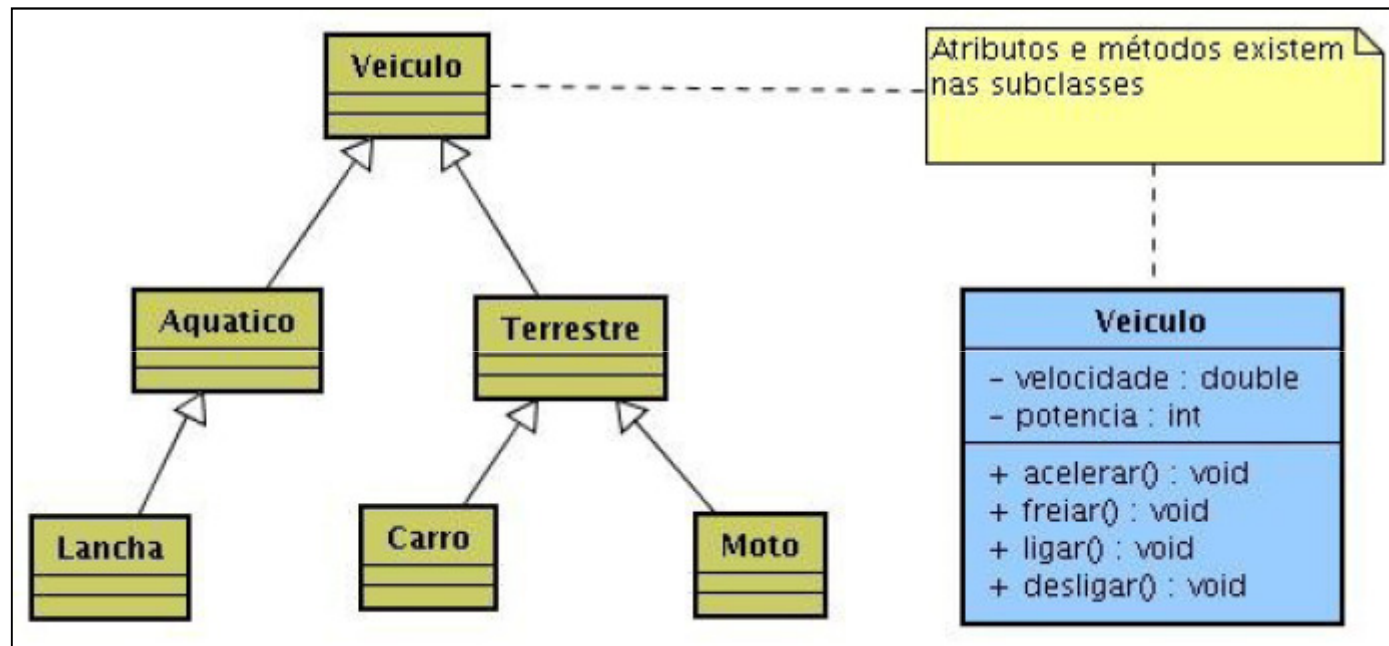
# Revisando - Conceitos

- Reuso de uma classe
  - Tudo que foi definido para uma classe vai ser aproveitado em outra classe
- Representa especialização
  - Um tipo mais genérico cujas características serão herdadas por outra classe
  - Comportamento pode ser herdado ou modificado
- Classe A e classe B
  - B herda de A
  - B é subtipo de A
  - B é subclasse A
- Todos os atributos e métodos definidos em A também existem em B

# Herança



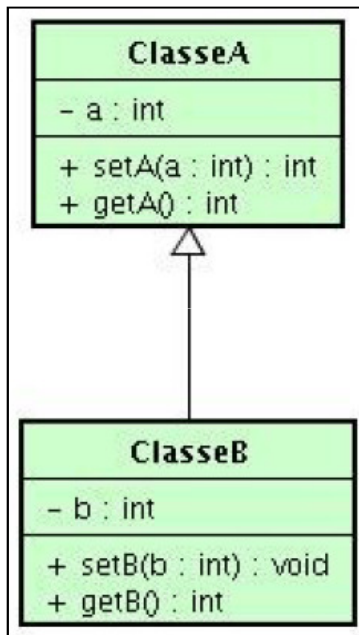
# Herança



Carro possui todas as características de Terrestre e também de Veiculo



# Herança - Implementação



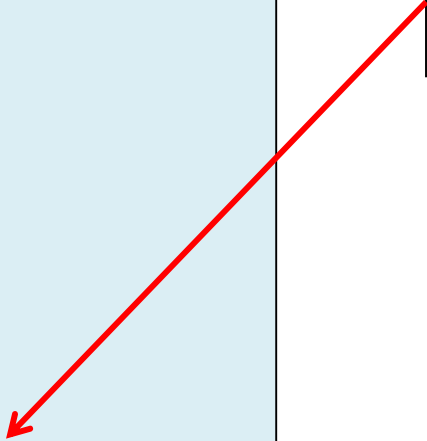
**extends é usado para  
indicar herança em JAVA**

```
class ClasseA {
    protected int a;
    public int getA() {
        return a;
    }
    public void setA(int a) {
        this.a = a;
    }
}
class ClasseB extends ClasseA{
    private int b;
    public int getB() {
        return b;
    }
    public void setB(int b) {
        this.b = b;
    }
}
```

# Herança - Implementação

```
public static void main(String args[]) {  
    ClasseA a = new ClasseA();  
    ClasseB b = new ClasseB();  
    a.setA(10);  
    b.setA(20);  
    b.setB(30);  
    System.out.println(a.getA());  
    System.out.println(b.getB());  
    System.out.println(b.getA());  
}
```

Método herdado de  
ClasseA



Todos os membros definidos em ClasseA  
também existem em ClasseB

# Subtipos

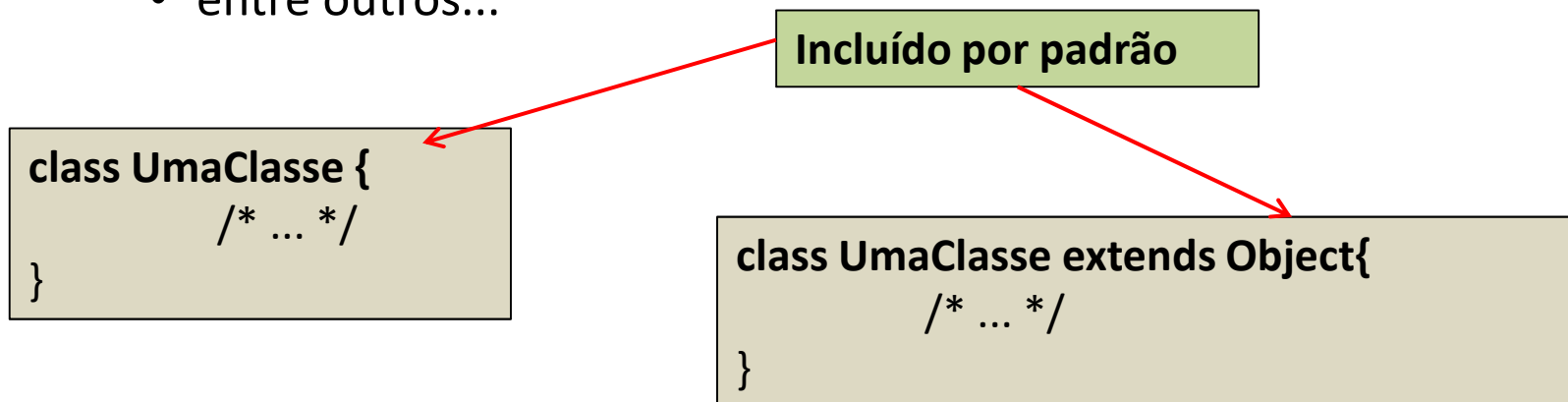
- Uma classe que herda de outra é um subtipo
  - Herança representa relacionamento “é um”
    - Carro “é um” veículo
    - Livro “é uma” publicação
    - Cachorro “é um” mamífero
- Podemos ter uma variável do tipo ClasseA que referencia um objeto do tipo ClasseB

**variável b NÃO  
executa métodos  
de ClasseB, pois  
ela só conhece a  
interface de  
ClasseA**

```
ClasseA a,b;  
a = new ClasseA();  
b = new ClasseB();  
a.setA(10);  
b.setA(20);  
//b.setB(30);  
System.out.println(a.getA()+b.getA());
```

# Classe Object

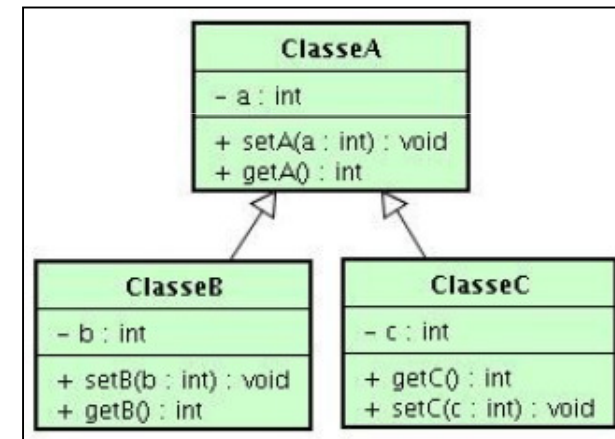
- Classe base em JAVA
  - Tudo herda de **Object**
  - Define alguns métodos:
    - public String toString()
    - public boolean equals()
    - entre outros...



# Teste de Igualdade

- Operador instanceof
  - Determina se um objeto obj é de uma classe Cla
    - retorna valor lógico (booleano)
    - sintaxe: obj instanceof Cla

```
public static void main(String args[]){
    ClasseA obj;
    obj = new ClasseC();
    if (obj instanceof ClasseA)
        System.out.println("obj é ClasseA");
    if (obj instanceof ClasseB)
        System.out.println("obj é ClasseB");
    if (obj instanceof ClasseC)
        System.out.println("obj é ClasseC");
}
```



obj é da *ClasseA*  
e da *ClasseC*

# Construtores

- Toda classe JAVA tem pelo menos um construtor
  - Se não for definido cria um padrão
    - `public NomeClasse(){...}`
- Todo construtor **deve chamar o construtor da superclasse**
  - Deve ser o primeiro comando do construtor
  - Compilador coloca código caso o programador não coloque

```
class ClasseD extends ClasseA{  
    public ClasseD(){  
        super();  
        /* ... */  
    }  
}
```

```
class ClasseD extends ClasseA{  
    public ClasseD(){  
        /* ... */  
    }  
}
```

# Construtores

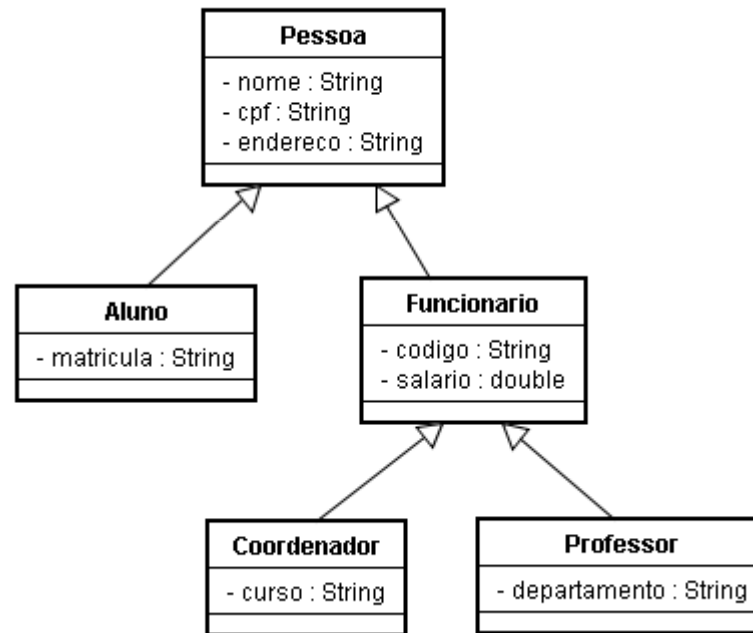
- O código abaixo pode não compilar

```
class ClasseD extends ClasseA{  
    public ClasseD(){  
        /*...*/  
    }  
}
```

- Se ClasseA não possuir construtor sem argumentos

# Exercício em Sala

- Implementar as classes do seguinte diagrama:



- Detalhes no próximo slide



# Exercício em Sala

- Todas as classes devem ter construtor parametrizado, acessando os construtores da classe pai
- Criem uma classe de teste