# MÉTODO CONSTRUTOR E SOBRECARGA

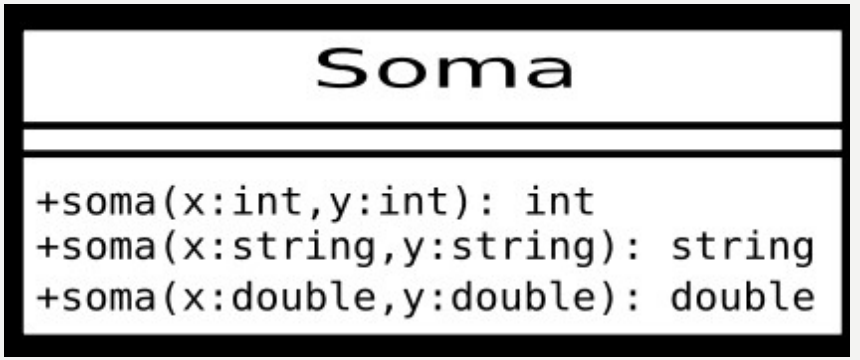
# MÉTODO CONSTRUTOR

- Quando um objeto é criado ou instanciado (alocado na memória através do operador new), um método definido na classe desse objeto sempre é automaticamente chamado, esse método é designado de construtor.
- Ele SEMPRE deve ter o mesmo nome da classe e não retornar valor.
- O método construtor é criado para inicializar os atributos do objeto e assim ter consistencia nos dados.
- É comum o desenvolvimento de mais de um método construtor com o mesmo nome, porém, com parâmetros diferentes, que são denominados construtores sobrecarregados.

```
public class Carro{

private String cor;
private double preco;
private String modelo;

  /* CONSTRUTOR PADRÃO */
  public Carro(){

  }
}
```

# SOBRECARGA DE MÉTODO

- Permite a existência de vários métodos de mesmo nome, porém com assinaturas levemente diferentes ou seja variando no número , tipo de argumentos , no valor de retorno e até variáveis diferentes.
- A sobrecarga de método dá mais opções ao programador na hora de programar a chamada de métodos de uma classe.



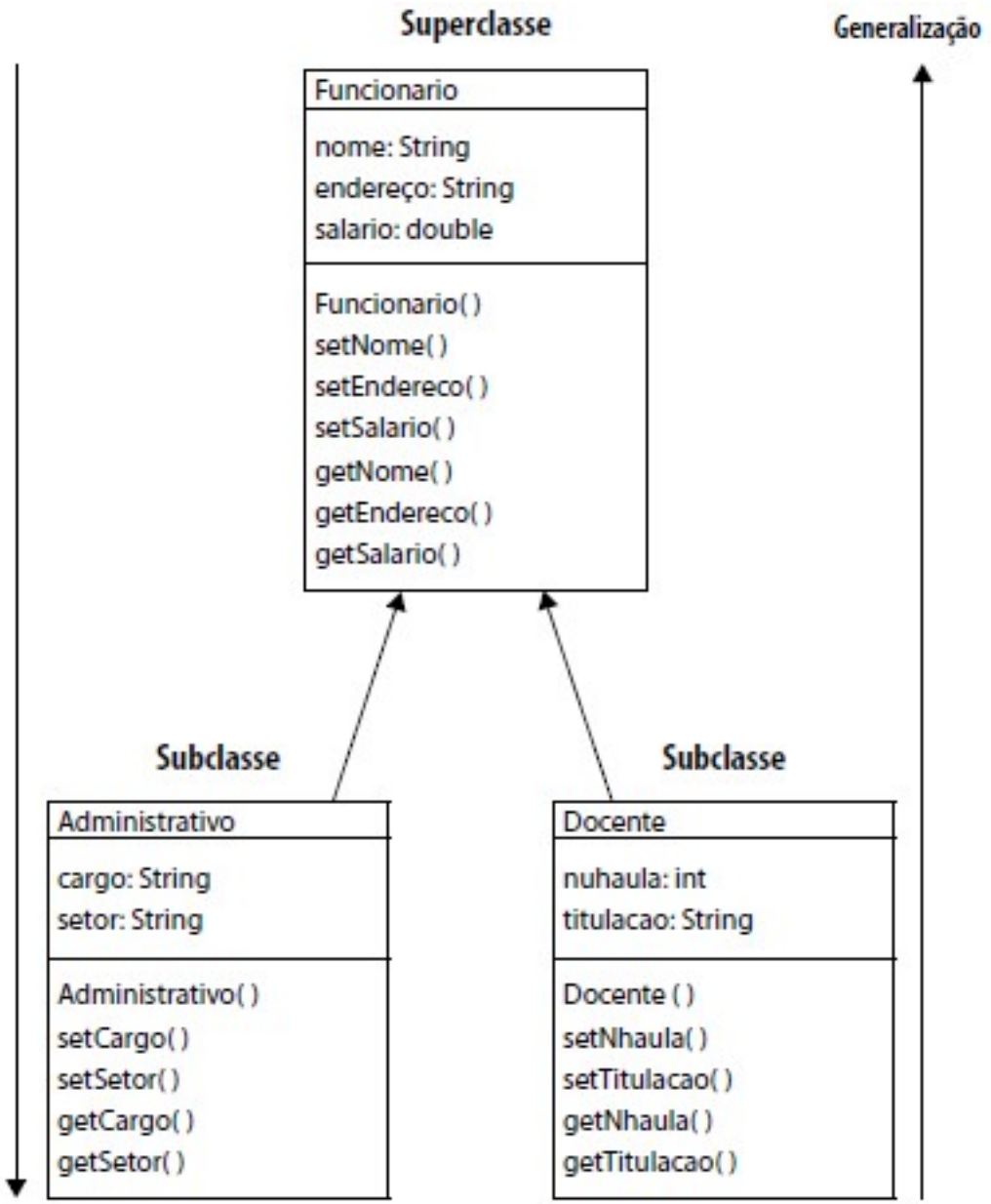
```java
public class Carro{

private String cor;
private double preco;
private String modelo;

/* CONSTRUTOR PADRÃO */
public Carro(){

}

/* CONSTRUTOR COM 2 PARÂMETROS */
public Carro(String modelo, double preco){
   //Se for escolhido o construtor sem a COR do veículo
   // definimos a cor padrão como sendo PRETA
   this.cor = “PRETA”;
   this.modelo = modelo;
   this.preco = preco;
}

/* CONSTRUTOR COM 3 PARÂMETROS */
public Carro(String cor, String modelo, double preco){
   this.cor = cor;
   this.modelo = modelo;
   this.preco = preco;
}

}
```

# HERANÇA

DISCIPLINA: PARADIGMAS ORIENTADA A OBJETOS

A Herança é um recurso da orientação a objetos que permite que **atributos e comportamentos (métodos) COMUNS** a diversos tipos de objetos, existentes em um problema, sejam agrupados e representados em uma única classe base, conhecida como **SUPERCLASSE.**

Os **atributos e comportamentos (métodos) ESPECÍFICOS** de cada tipo de objeto presente no problema são representados por classes específicas desses tipos de objetos. Essas classes específicas, conhecidas como **SUBCLASSES, herdam** os atributos e comportamentos comuns que foram agrupados na superclasse.


Superclasse
Generalização
Funcionario
nome: String
endereço: String
salario: double
Funcionario( )
setNome( )
setEndereco( )
setSalario( )
getNome( )
getEndereco( )
getSalario( )
Subclasse
Subclasse
Administrativo
cargo: String
setor: String
Administrativo( )
setCargo( )
setSetor( )
getCargo( )
getSetor( )
Docente
nuhaula: int
titulacao: String
Docente ( )
setNhaula( )
setTitulacao( )
getNhaula( )
getTitulacao( )

- A partir de uma classe base (superclasse), outras classes podem ser **especificadas** ou **especializadas**.
  - Uma subclasse é uma **especialização** de uma superclasse.
  - Uma superclasse é uma **generalização** de uma subclasse.

# OBJETIVO DA HERANÇA

- O objetivo principal da herança é a **reutilização de código**, já que novas classes (subclasses) podem ser criadas a partir de outra já existente (superclasse), herdando seus atributos e métodos.
- A reutilização de código economiza tempo de desenvolvimento de programas.

# NÍVEIS DE RELACIONAMENTO DE HERANÇA

- Chama-se **superclasse direta** de uma subclasse, aquela imediatamente superior a essa subclasse e, **superclasse indireta,** aquela de dois ou mais níveis acima da hierarquia.
- Por exemplo, uma classe Docente herda de Funcionário que poderia herdar de Pessoa.
  - Desta forma, Funcionário é superclasse direta de Docente e Pessoa seria superclasse indireta de Docente.

# NÍVEIS DE RELACIONAMENTO DE HERANÇA

- Ao se instanciar um objeto de qualquer subclasse, pode-se acessar diretamente os membros com qualificador de acesso public da superclasse (normalmente métodos) como se fossem parte da subclasse.
- Isso só é possível por causa do relacionamento de herança que existe entre a superclasse e a subclasse.

# SINTAXE

- A palavra-chave **extends** indica que uma subclasse estende a superclasse, ou seja, ela herda.

```
public class Administrativo extends Funcionario{
        private String cargo, setor;
```

- Com a representação acima a subclasse Administrativo estende a superclasse Funcionário.

# SINTAXE

```
public Administrativo ( ){
  super( );
  cargo = "";
  setor = "";
}
```

- Essa instrução **super( )** chama explicitamente o método construtor da superclasse.
- Sempre deve estar programada na primeira linha.
- Quando o método construtor de Administrativo for chamado (quando se cria um objeto do tipo Administrativo) a primeira instrução é chamar o método construtor da superclasse Funcionário.

```
public class Administrativo extends Funcionario{
    private String cargo, setor;

    public Administrativo ( ){
      super( );
      cargo = "";
      setor = "";
    }

    public void setCargo(String scargo){
      cargo=scargo;
    }
```

```
public class Docente extends Funcionario{
        private int nha;
        private String titulacao;

        public Docente(){
          super( );//chamada explicita ao construtor
da superclasse
          nha=0;
          titulacao="";
        }
```

# MÉTODO CONSTRUTOR EM SUBCLASSE

- Na implementação do método construtor de uma subclasse o construtor da superclasse deve ser explicitamente chamado na primeira linha através da instrução super( ).

```
Linha 4
Linha 5          public Docente(){
Linha 6            super( );//chamada explícita ao construtor
                da superclasse
Linha 7            nha=0;
Linha 8            titulacao="";
Linha 9          }
```

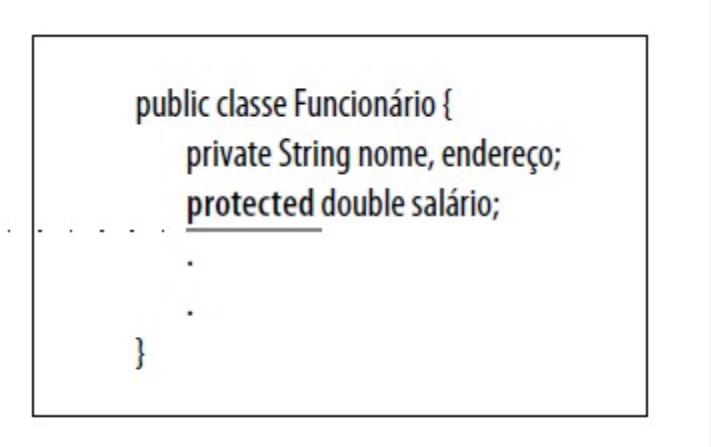
# MODIFICADOR DE ACESSO PUBLIC E PRIVATE

- Os membros (atributos e métodos) **public** de uma superclasse podem ser acessados diretamente (somente pelo nome) em todas as subclasses que estendem essa superclasse.

- Os membros (atributos e métodos) **private** de uma superclasse somente são acessados pelos métodos dessa superclasse, ou seja, a subclasse não pode acessar diretamente os membros **private** da sua superclasse.

# MODIFICADOR DE ACESSO PROTECTED

- O modificador **protected** pode ser utilizado na definição de atributos e métodos da superclasse.
- Quando atributos e métodos da superclasse são definidos com o modificador protected, eles são "liberados" para acesso direto (pelo nome) pelas subclasses que estendem essa superclasse.



# REDEFINIÇÃO DE MÉTODOS (MÉTODO SOBRESCRITO)

- Outro conceito importante ligado à herança é o de **método sobrescrito** ou **método redefinido**.
- Isso acontece quando uma subclasse (que herda todos os métodos da sua superclasse) redefine ou sobrescreve (**override**) um método herdado da sua superclasse.
- Redefinir ou sobrescrever um método herdado na subclasse é implementar esse método herdado NOVAMENTE utilizando a mesma assinatura.
- **Assinatura do método** é o cabeçalho do método, ou seja, modificador + tipo de retorno + nome + parâmetros.
- A sobrescrita de método é necessária quando a subclasse não deseja herdar um determinado método da maneira como ele foi implementado.

# REDEFINIÇÃO DE MÉTODOS (MÉTODO SOBRESCRITO)