**Anotações POO**

💡 Na programação orientada a objetos (POO), uma classe é uma estrutura fundamental que atua como um modelo para criar objetos.

**Classe:**

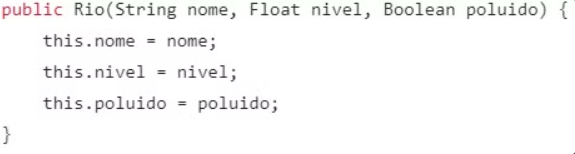
Uma classe é como um projeto ou uma planta baixa que define como os objetos serão construídos.

Características -> Possuí atributos e métodos

**Construtores:**

—> São métodos especiais utilizados para inicializar objetos quando eles são criados. Permitem a definição de como os objetos devem ser construídos. Um construtor tem o mesmo nome da classe e pode ter diferentes assinaturas.

—> São eles que atribuem o valor informado pelo usuário para o objeto.



—> Para criar um objeto de alguma classe, por exemplo essa acima, precisamos passar os “parâmetros”: nome, nivel e poluido.

**Encapsulamento**

O encapsulamento é um dos princípios fundamentais da programação orientada a objetos (POO) e refere-se à prática de esconder os detalhes internos de um objeto e fornecer uma interface pública para interagir com esse objeto.

—> Getters e Setters controlam o acesso aos atributos.

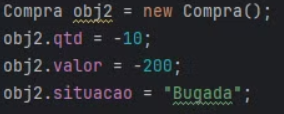
**Em Java, existem quatro tipos de modificadores de acesso**

* O modificador **public** permite que um membro (método, atributo, classe) seja acessado por qualquer outra classe.
* O modificador **private** restringe o acesso ao membro apenas à classe em que está definido.
* O modificador **protected** permite acesso aos membros dentro da mesma classe, subclasses e classes no mesmo pacote.
* O modificador **default** (ausência de modificador) restringe o acesso ao membro apenas para classes no mesmo pacote.

**Vantagens**

—> Controle de dados e como eles são utilizados, facilitando manutenção e refatoração de código.

Variáveis publicas podem serem alteradas dessa forma

****

**—> Na variável private isso não acontece (mesmo se o dado passado estiver certo)**

**BENEFÍCIOS**

* Protege os detalhes internos da implementação de uma classe, reduzindo o acoplamento entre classes e facilitando a manutenção e a evolução do código.
* Facilita a reutilização de código, pois permite que os detalhes de implementação sejam alterados sem afetar outras partes do programa que dependem da classe encapsulada.
* Melhora a segurança do código, pois restringe o acesso direto aos dados da classe, permitindo que a classe controle como seus dados são manipulados e validados

**Associação entre Classes**

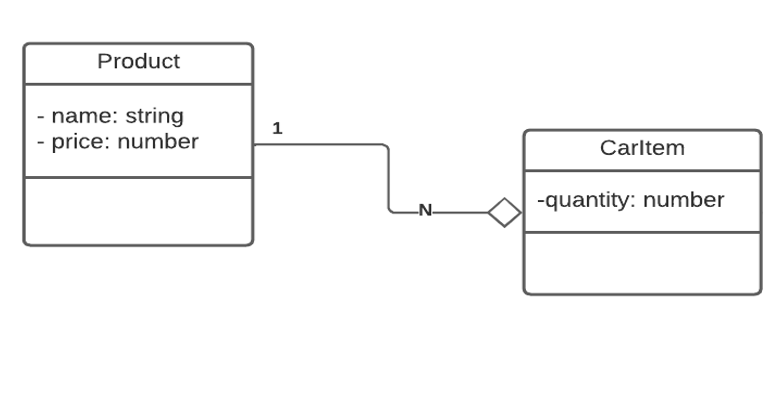
Em uma relação de associação, uma classe é o todo e a outra classe é a parte

**Tipos de associação**

* + Agregação
  + Composição

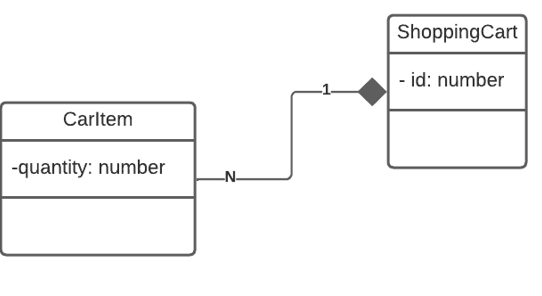
**Na agregação**

* No diagrama de classe: diamante não preenchido no lado da classe **todo**
* a **parte** não depende do **topo**, são independentes,
* se o **todo** for eliminado, a **parte** não será
* **parte** agrega o **todo**
* É uma associação fraca
* Toda vez que a cardinalidade for N do lado do diamente, temos uma agregação



**Na composição**

* No diagrama de classe, diamante preenchido
* a classe parte depende do todo, são dependentes
* se o todo for eliminado, a parte será também
* parte está contida no todo
* É uma associação forte
* Toda vez que a cardinalidade for 1 do lado do diamente, temos uma composição

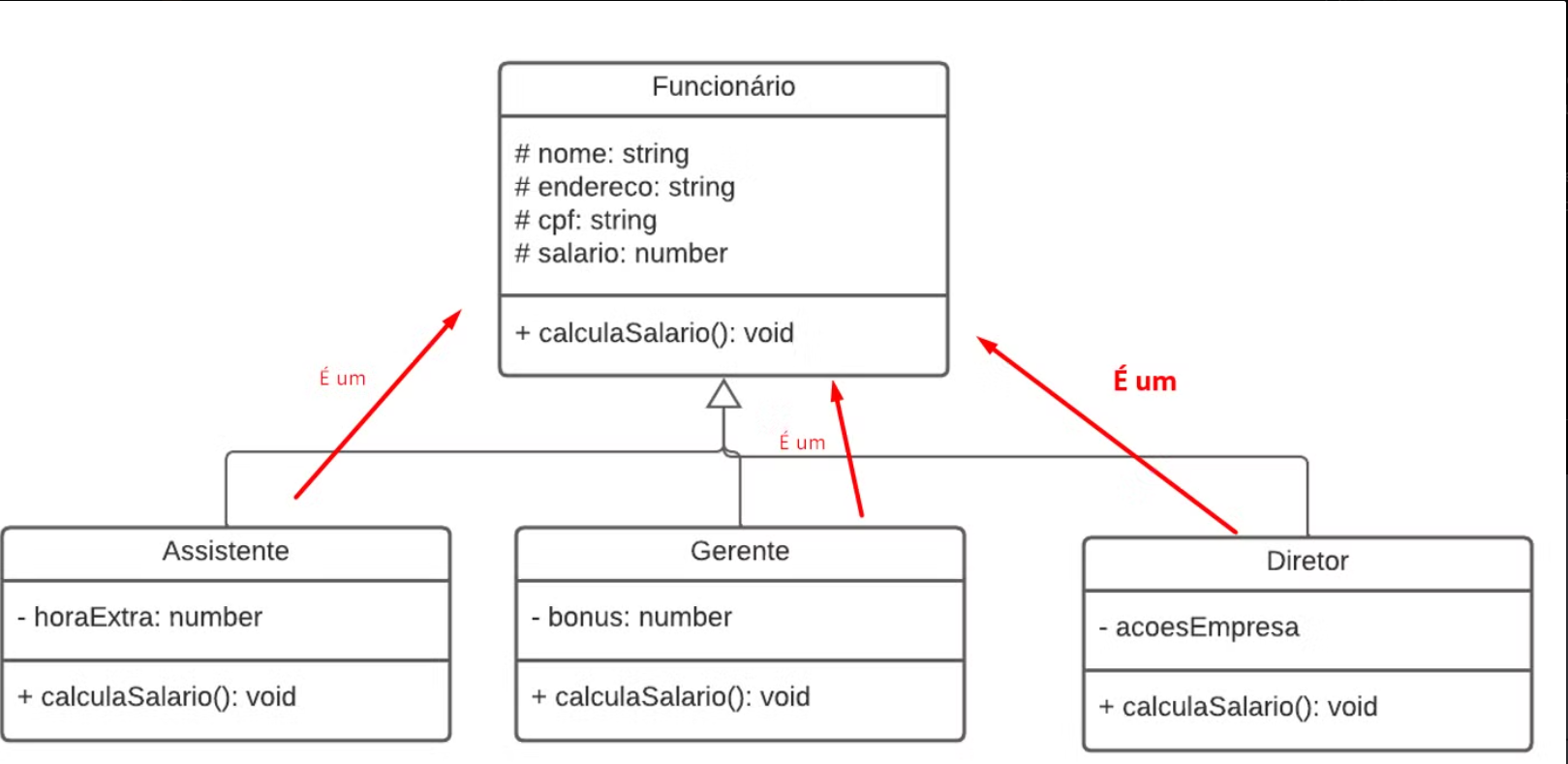


**Classes no conceito de herança**

* **Classe Base (ou Superclasse)**: Também conhecida como classe pai ou classe mãe, é a classe da qual outra classe herda. Ela contém os atributos e métodos comuns que são compartilhados pelas subclasses.
* **Classe Derivada (ou Subclasse)**: Também chamada de classe filha, é a classe que herda de outra classe. Ela pode adicionar novos atributos e métodos ou modificar os existentes, além de herdar os atributos e métodos da classe base.

**Agreg. e Compos. VS Herança**

* No conceito de herança, podemos imaginar um diagrama "é um...",
* No conceito de agreg. e compos. a pergunta de associação "é um..." sempre será falso.



* + ⇒ Visibilidade privada
  + ⇒ Visibilidade pública

**⇒ Visibilidade protegida**

* + Protegida é publica para as filhas, porém é privada para qualquer outra classe.