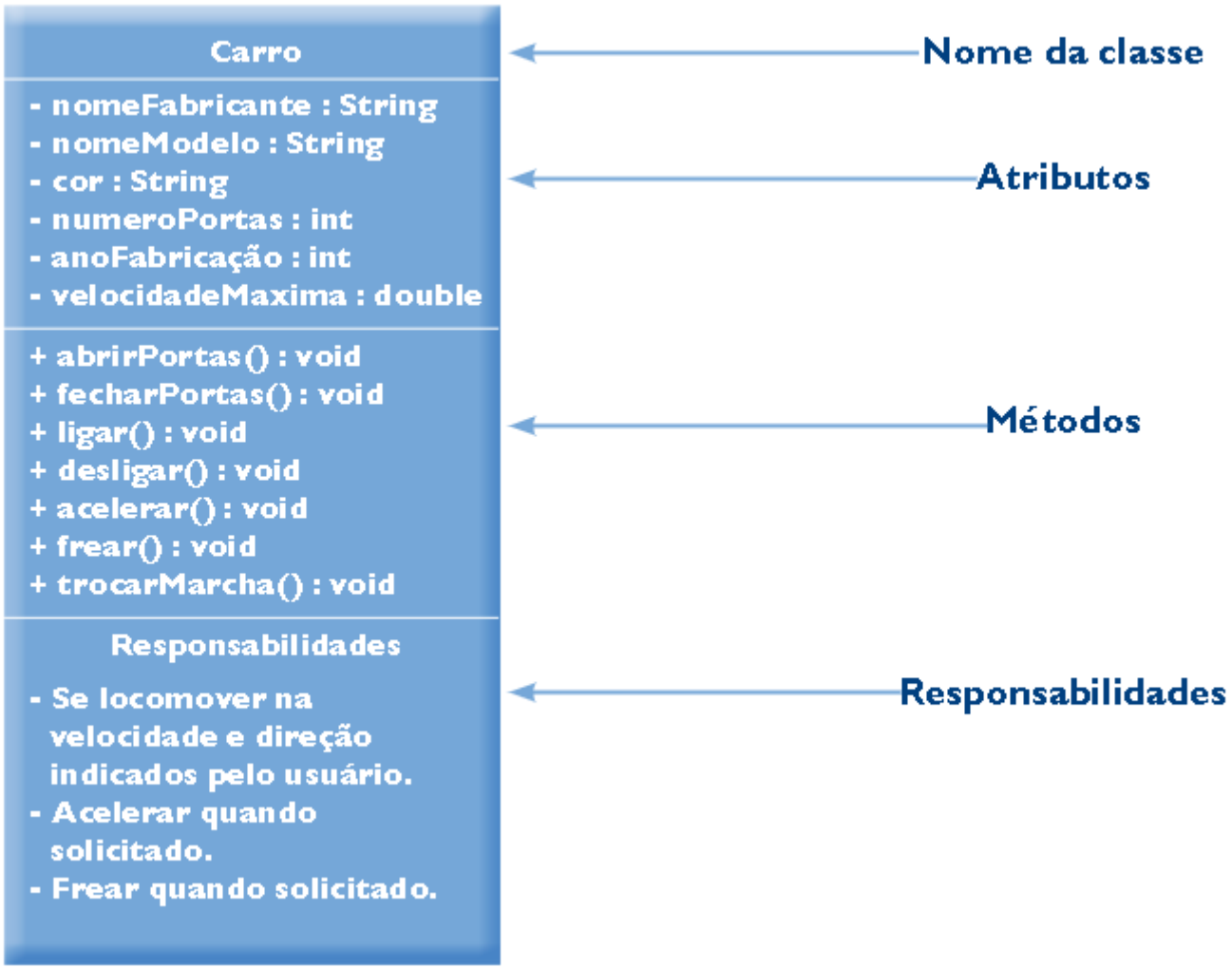
Diagrama de Classes

Visão Lógica

Classe

- De acordo com Booch, Rumbaugh e Jacobson (2005), classe é uma descrição de um conjunto de objetos que compartilham os mesmos atributos, operações, relacionamentos e semântica. A representação completa de uma classe tem quatro divisões.

- Nome da classe – Cada classe deve ter um nome que a diferencie das outras classes (BOOCH, RUMBAUGH e JACOBSON, 2005).
- Atributo – É uma propriedade nomeada de uma classe, que descreve um inter-valo de valores que as instâncias da propriedade podem apresentar (BOOCH, RUMBAUGH e JACOBSON, 2005).


Carro
- nomeFabricante : String
- nomeModelo : String
- cor : String
- numeroPortas : int
- anoFabricação : int
- velocidadeMaxima : double
+ abrirPortas() : void
+ fecharPortas() : void
+ ligar() : void
+ desligar() : void
+ acelerar() : void
+ frear() : void
+ trocarMarcha() : void
Responsabilidades
- Se locomover na velocidade e direção indicados pelo usuário.
- Acelerar quando solicitado.
- Frear quando solicitado.
Nome da classe
Atributos
Métodos
Responsabilidades

Método

- É a implementação de um serviço que pode ser solicitado por um objeto da classe para modifi car o seu comportamento, algo que pode ser feito com um objeto e que é compartilhado por todos os objetos dessa classe (BOOCH, RUMBAUGH e JACOBSON, 2005). Existem alguns métodos especiais em praticamente todas as classes, os quais, geralmente, não representamos nos diagramas da UML por já terem se tornado senso comum entre os desenvolvedores. Mas, sempre que você achar necessário, poderá defi ni-los dentro da classe.

Os métodos especiais

- Construtor: é o método que constrói, isto é, reserva o espaço em memória onde serão armazenadas as informações daquele objeto da classe.
- Get: é o método que apresenta o valor armazenado em determinado atributo de um objeto.
- Set: dá um valor a um atributo.

Os métodos especiais

- Os métodos get e set são muito úteis para preservar os atributos e garantir que sua alteração seja feita unicamente por intermédio deles. Chamamos isso de encapsulamento dos atributos de uma classe, pois podemos deixar todos eles com visibilidade privada e só manipulá-los utilizando os métodos get (para retornar o valor que está no atributo) e set (para atribuir um valor a ele). Geralmente adotamos um método set e um método get para cada atributo da classe.

Os métodos especiais

- Destrutor: destrói o objeto criado da memória, liberando o espaço de memória alocado na sua criação. Não é necessário criá-lo em linguagens orientadas a objetos que possuam garbage collector, isto é, que excluam os objetos que já não tenham referência alguma na memória.

Assinatura

- É a primeira linha da definição de um método, no qual podemos observar sua visibilidade, seu nome, seus parâmetros de entrada e de retorno.

- Exemplo:

+ soma(valor1:double,valor2:double):double

Assinatura

- o símbolo + no início da assinatura demonstra que se trata de um método público, ou seja, que todos os objetos que tiverem acesso à classe a que pertencem também poderão acessá-lo;
- o nome do método é soma;
- o método soma recebe como parâmetros de entrada dois valores do tipo double, chamados valor1 e valor2, e devolve como resultado um número do tipo double.

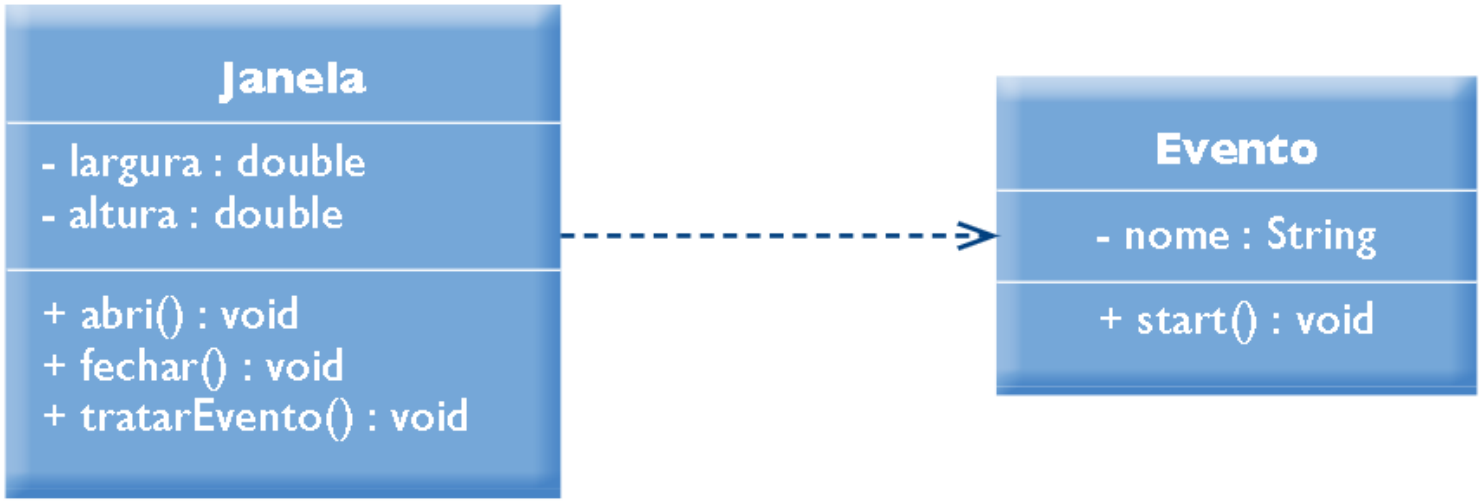
Responsabilidades

- São contratos ou obrigações de determinada classe. Ao criarmos uma classe, estamos criando uma declaração de que todos os seus objetos têm o mesmo tipo de estado e o mesmo tipo de comportamento (BOOCH, RUMBAUGH e JACOBSON, 2005).
- Dependendo do nível de detalhe (abstração) que estamos analisando no diagrama, podemos também representar graficamente uma classe apenas com seu nome ou com nome dos principais atributos e principais métodos, conforme o que queremos analisar no momento em que estamos criando o diagrama.
- Não precisamos, então, escrever todos os componentes da classe.

Tipos de relacionamento entre classes

- Existem basicamente três tipos principais de relacionamento entre classes: dependência, associação e herança.
- **Dependência**: é um relacionamento de utilização, determinando que um objeto de uma classe use informações e serviços de um objeto de outra classe, mas não necessariamente o inverso. A dependência é representada graficamente por uma linha tracejada com uma seta indicando o sentido da dependência.

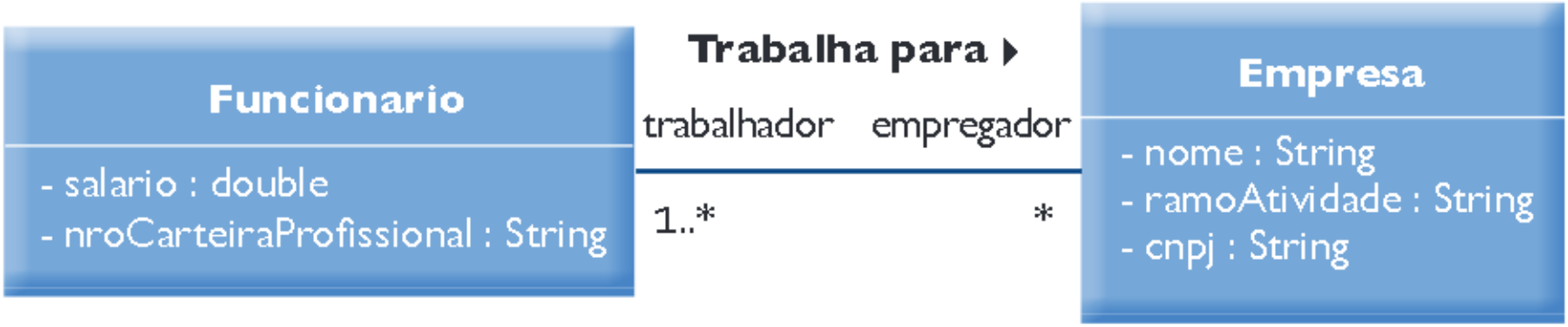
Dependência



Tipos de relacionamento entre classes

- **Associação**: é um relacionamento estrutural que especifica objetos de uma classe conectados a objetos de outra classe. A partir de uma associação, conectando duas classes, você é capaz de navegar do objeto de uma classe até o objeto de outra classe e vice-versa (BOOCH, RUMBAUGH e JACOBSON, 2005). Representada por uma linha interligando as duas classes, uma associação pode definir papéis das classes relacionadas, assim como a multiplicidade de sua associação, além de ter um nome. Mas nenhum desses componentes é obrigatório em uma associação e só devem ser usados para deixar mais clara a sua definição.

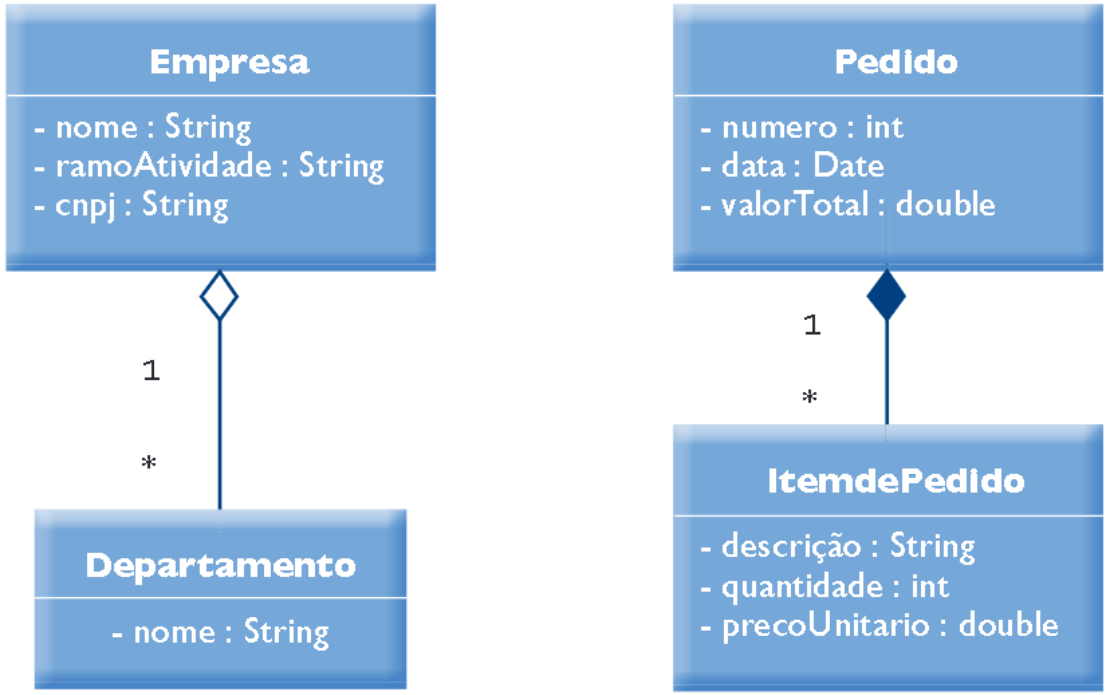
Associação



- No diagrama da figura, identificamos uma **associação**, representada em UML 2.0 por uma linha interligando as duas classes, de nome Trabalha para, onde a classe Funcionario representa o papel de trabalhador e a classe Empresa representa o papel de empregador. Podemos observar pela representação da multiplicidade que cada objeto da classe Empresa tem, como trabalhador, 1 ou mais objetos da classe Funcionário e que um objeto da classe Funcionario tem, como empregador, no mínimo 0 e no máximo vários objetos empresa.

Tipos de relacionamento entre classes

- Já a agregação é um tipo de associação entre classes na qual é mostrada a relação todo/parte, nela uma classe fará o papel do todo e a outra, da parte. A agregação entre duas classes é representada em UML 2.0 como uma linha ligando duas classes com um losango na ponta da classe toda para diferenciar o todo da parte.
- A composição, por sua vez, é uma forma de agregação com propriedade bem definida e tempo de vida coincidente das partes pelo todo. As partes com multiplicidade não associada poderão ser criadas após a própria composição, mas, uma vez criadas, vivem e morrem com ela. Estas partes só podem ser removidas explicitamente antes da morte do elemento composto (BOOCH, RUMBAU-GH e JACOBSON, 2005).



Observamos nesse diagrama da figura, que uma empresa é formada por um conjunto de departamentos, ou seja, tem a multiplicidade.

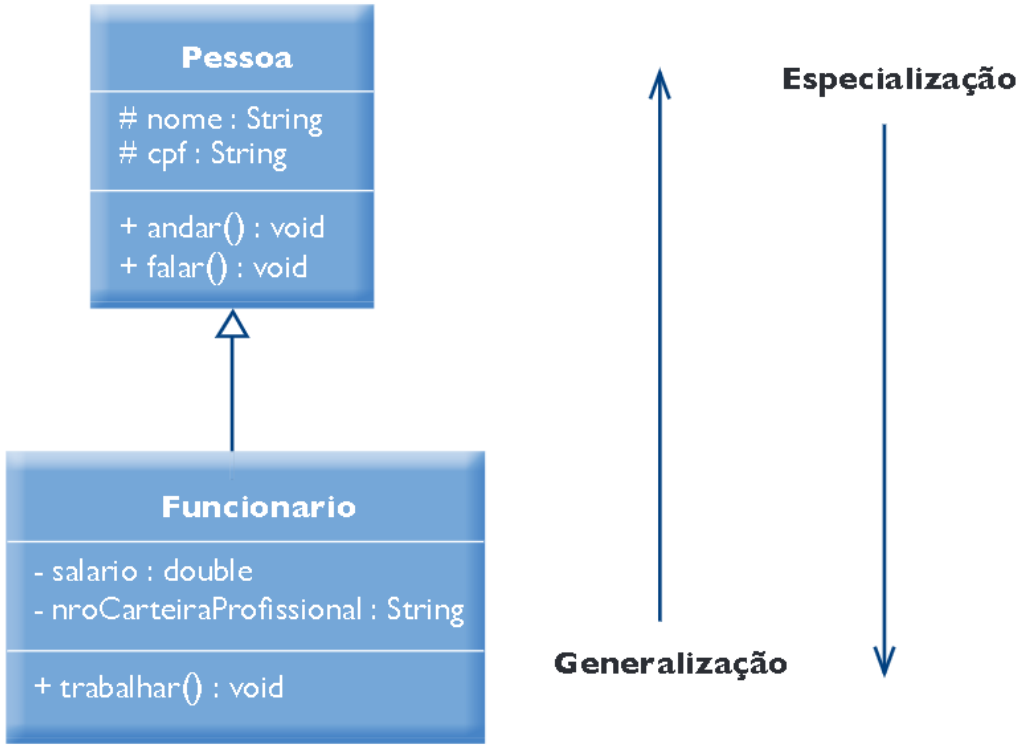
Podemos dizer que numa relação de composição só faz sentido existir a parte se houver o todo.

Podemos analisar que só faz sentido existirem os itens de pedido (parte) se existir o pedido (todo).

- Em relação à multiplicidade podemos ter:
  - 0..* = multiplicidade de zero a muitos.
  - 1..* = multiplicidade de 1 a muitos.
  - 0..1 = multiplicidade de 0 a 1.
  - * = multiplicidade de zero a muitos; possui o mesmo signifi cado de 0..*.
  - n = multiplicidade n, onde n deve ser trocado por um número que representará o número exato de objetos relacionados àquela classe.

Tipos de relacionamento entre classes

- **Herança**: refere-se ao mecanismo pelo qual classes mais específicas incorporam a estrutura e o comportamento de classes mais gerais (BOOCH, RUM-BAUGH e JACOBSON, 2005).



- Podemos verificar, na figura, que a classe Pessoa possui os atributos nome e cpf e pode executar os métodos de andar e falar. Já a classe Funcionario, por herdar os atributos e métodos da classe Pessoa, possui nome, cpf, salário e nro Carteira Profissional, podendo executar os métodos andar, falar e trabalhar. Dizemos que a classe Pessoa é a superclasse da classe Funcionario, que é sua subclasse, ou que Pessoa é a classe pai de Funcionario e que Funcionario é classe filha de Pessoa.

Objeto

- É uma manifestação concreta de uma abstração; uma entidade com uma fronteira bem definida e uma identidade que encapsula estado e comportamento; instância de uma classe (BOOCH, RUMBAUGH e JACOBSON, 2005). Podemos imaginar uma classe como sendo o molde e os objetos, os produtos criados a partir desse molde. Um bom exemplo é pensar nos atributos do carro como sendo: modelo, número de portas, cor, ano de fabricação, tipo de combustível, velocidade máxima. Já os métodos do carro podemos definir como: andar, parar, acelerar, entre outros. Pensando em responsabilidades, podemos dizer que o carro tem a responsabilidade de obedecer aos comandos de seu piloto, conduzindo-o na velocidade e pelo caminho que ele escolheu, acelerando e freando de acordo com o que foi sinalizado.

Interação entre objetos

- Agora que nós já definimos e diferenciamos classes e objetos, precisamos saber como os objetos interagem entre si. De acordo com o paradigma de orientação a objetos, isso ocorre por meio de trocas de mensagens. Para entendermos como essas trocas funcionam, estudemos mais alguns conceitos.

Mensagem

- Segundo Booch, Rumbaugh e Jacobson (2005), mensagens são a especificação de uma comunicação entre objetos que contêm informações à espera da atividade que acontecerá, isto é, as informações trocadas entre os objetos para que executem as funções necessárias para o funcionamento do sistema.

Outros conceitos

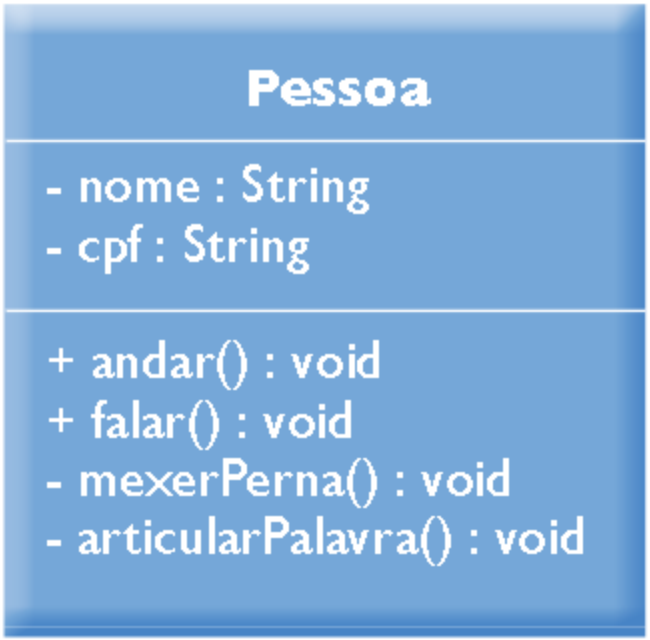
- **Encapsulamento**: Propriedade de uma classe de restringir o acesso a seus atributos e métodos. A possibilidade de definir áreas públicas e privadas em sua implementação garante mais segurança ao código criado, já que podemos definir os parâmetros de entrada e saída de um método sem revelar como ele é implementado.

Outros conceitos

- **Visibilidade**: Existem quatro tipos de visibilidade de atributos ou métodos:
  - Private (privada): representada por um sinal de menos (–), é a visibilidade mais restrita e permite o acesso ao atributo ou método apenas dentro da própria classe.
  - Protected (protegida): representada por um sustenido (#), permite o acesso aos métodos e atributos dentro da própria classe ou de suas subclasses.
  - Public (pública): representada por um sinal de mais (+), permite o acesso aos métodos e atributos a todas as classes que tiverem algum tipo de relação com a classe em que foram criados. É a menos restritiva das visibilidades.
  - Package (pacote): representada por um til (~) permite o acesso aos métodos e atributos a todas as classes incluídas no mesmo pacote.

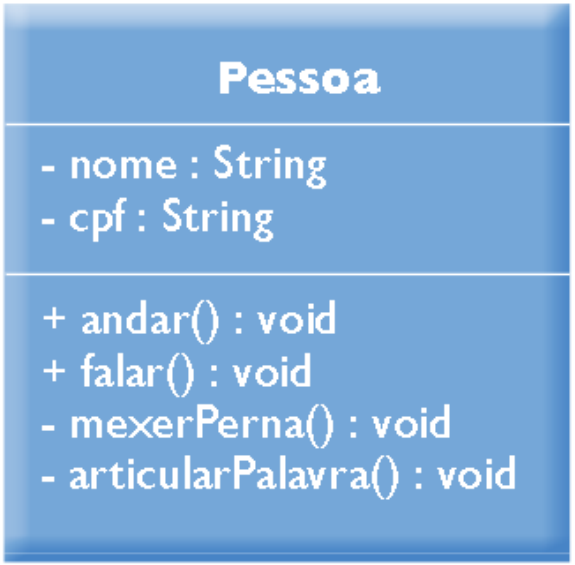
Métodos públicos e privados

- Se analisarmos a classe Pessoa, na figura, veremos que seus atributos são privados, mas como faremos para acessá-los? Utilizando os métodos get e set de cada atributo. É uma forma de garantir que os atributos somente serão alterados pelos métodos get e set, o que permite maior controle sobre seu uso.



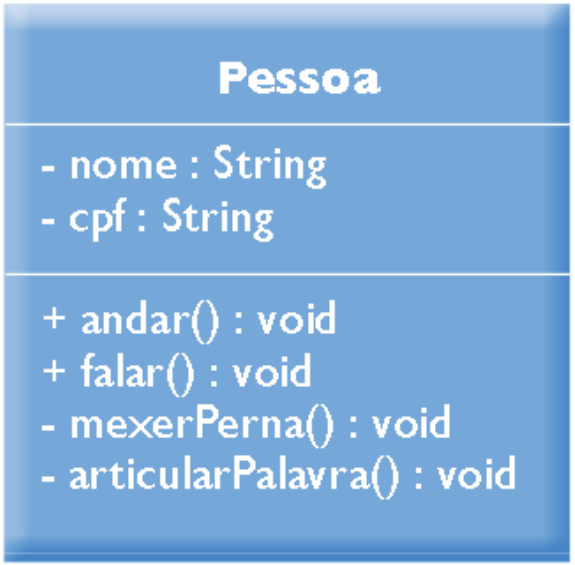
Métodos públicos e privados

Veja também que os métodos andar e falar são públicos, isto é, qualquer objeto que tiver acesso à classe poderá utilizá-los. Já os métodos mexerPerna e articularPalavra são privados, isto é, só podem ser utilizados dentro da classe Pessoa. Mas qual a vantagem desse tipo de implementação? Claro que para andar, uma pessoa tem que mexer a perna. O segredo do andar está no modo como ela mexe a perna. O que fizemos, então, foi garantir que o segredo, que também chamamos de regra de negócio, não faz parte da interface pública da classe, isto é, da forma com que as outras classes vão utilizá-la.



Métodos públicos e privados

Com o método encapsulado na classe, seu código está mais protegido de eventual cópia ou reprodução de como foi implementado, pois uma classe externa nem sabe de sua existência. Logo, da forma como projetamos sua implementação nenhuma outra classe saberá que para andar é necessário mexer a perna e muito menos como isso é feito. Essa é a vantagem do encapsulamento no desenvolvimento de software orientado a objetos. O mesmo princípio foi usado ao encapsular o método articularPalavra.

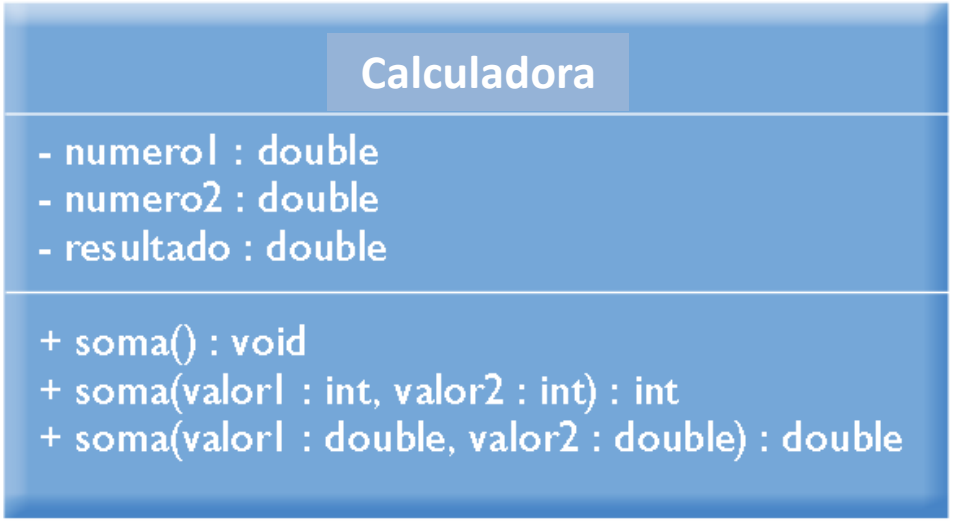


Polimorfismo

- Poliformismo: Palavra de origem grega que significa várias formas. No paradigma de orientação a objetos polimorfismo aparece em três formas diferentes:
- **Sobrecarga**: na mesma classe podemos ter métodos com o mesmo nome, mas que recebem parâmetros diferentes (assinaturas diferentes) e têm, por isso mesmo, funcionalidades e/ou implementações diferentes.

Sobrecarga

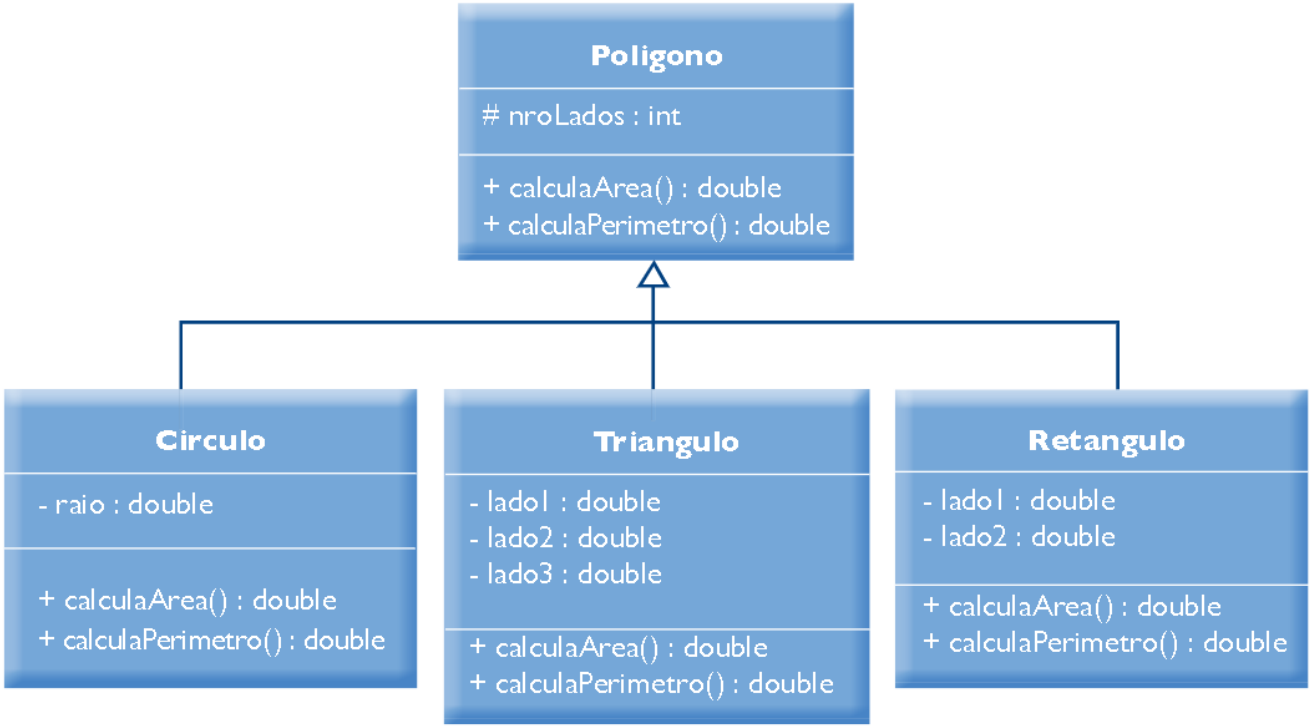
Neste exemplo, vemos que a classe Calculadora implementa três métodos soma diferentes: o primeiro efetua a soma com os valores dos atributos numero1 e numero2, colocando o resultado no atributo resultado. Já o segundo método soma recebe como parâmetros os números inteiros valor1 e valor2 e devolve como resultado a soma dos dois. O terceiro método recebe dois valores de tipo double valor1 e valor2 e retorna como resultado a sua soma. O interpretador de comandos escolherá, em tempo de execução, com base nos tipos e no número de parâmetros informados, qual dos métodos soma será executado.



Polimorfismo

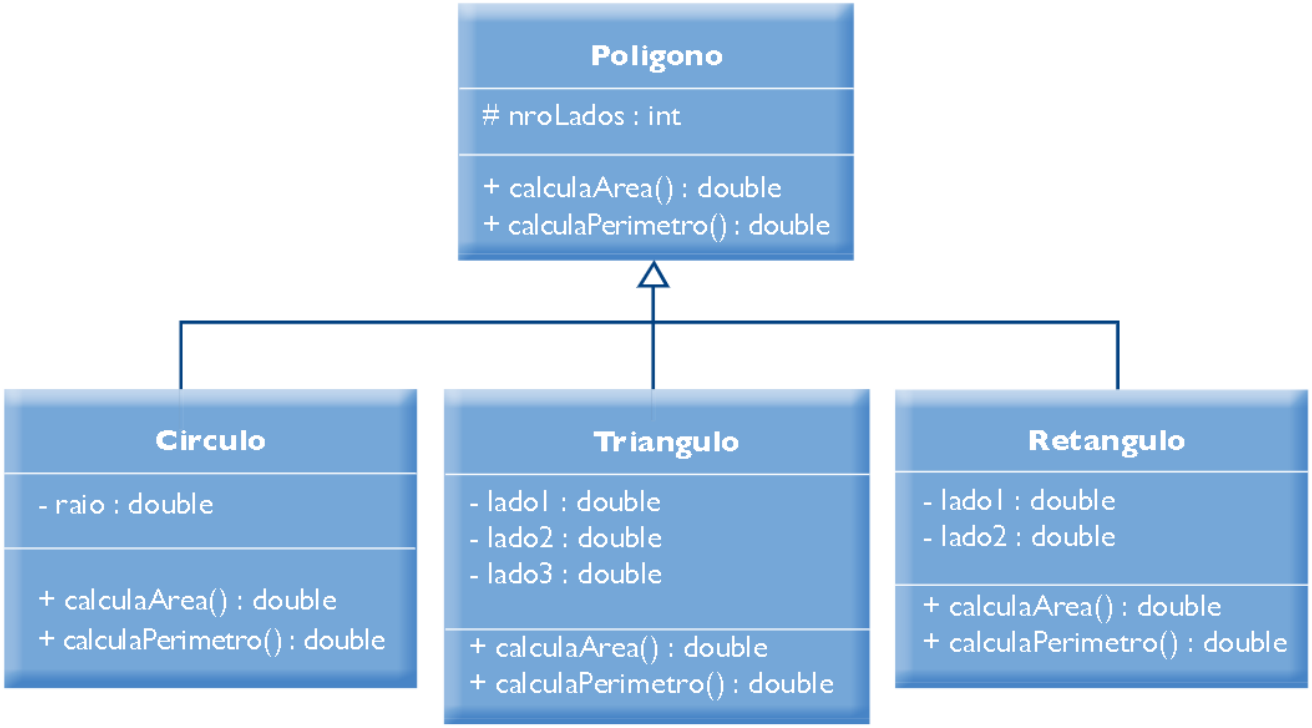
- **Sobrescrita**: em classes associadas por uma hierarquia pode haver métodos com a mesma assinatura, mas que efetuam operações diferentes. Assim, se optará pela execução de um ou de outro, dependendo da classe que estiver instanciada no momento da execução.

Sobrescrita



No exemplo há uma superclasse chamada Poligono, que implementa os cálculos de área e perímetro para os polígonos que não sejam círculo, triângulo e retângulo, para os quais foram criadas classes filhas, de acordo com suas fórmulas.

Sobrescrita



Esse é um exemplo de sobrescrita dos métodos calculaArea() e calculaPerimetro(), que, dependendo da classe a que pertence, o objeto instanciado usará o método implementado por essa classe.