A programação orientada a objetos (POO) é um paradigma de programação que se baseia no conceito de "objetos" para modelar e solucionar problemas do mundo real.

Em Java, uma linguagem de programação orientada a objetos, as classes são como plantas que definem a estrutura e o comportamento dos objetos. Imagine que você está construindo uma casa: a planta baixa seria a classe, e a casa em si, o objeto.

A planta define as características da casa, como número de quartos, tamanho da sala, etc., enquanto o objeto é a materialização daquela planta, a casa propriamente dita. Dentro de uma classe, definimos atributos e métodos.

Atributos são as características do objeto, como nome, idade, cor, etc. Já os métodos representam as ações que o objeto pode realizar, como andar, falar, calcular, etc. Voltando ao exemplo da casa, a cor das paredes seria um atributo, enquanto abrir a porta seria um método.

Para criar um objeto a partir de uma classe, utilizamos a palavra-chave "new", seguida do nome da classe e parênteses. Esse processo é chamado de instanciação. É como se estivéssemos construindo a casa de fato, utilizando a planta como guia. O encapsulamento é um dos pilares da POO e consiste em proteger os dados internos de um objeto, permitindo o acesso apenas através de métodos específicos. É como se tivéssemos uma caixa com ferramentas e, para usar cada ferramenta, precisamos usar a chave correta.

No contexto de classes, utilizamos modificadores de acesso, como "public", "private" e "protected", para controlar a visibilidade dos atributos e métodos. A herança é outro conceito fundamental da POO que permite criar novas classes a partir de classes já existentes, herdando seus atributos e métodos. É como se criássemos uma nova planta de casa a partir de uma planta base, adicionando ou modificando elementos. A classe que herda é chamada de subclasse, e a classe da qual ela herda é chamada de superclasse.

As interfaces, por sua vez, definem um contrato que as classes devem seguir. São como um conjunto de regras que as classes devem obedecer. Uma classe pode implementar múltiplas interfaces, o que garante que ela possua determinados métodos e comportamentos. As classes abstratas são classes que não podem ser instanciadas diretamente, servindo como modelo para outras classes. Elas podem conter métodos abstratos, que são métodos sem implementação, deixando a responsabilidade para as subclasses.

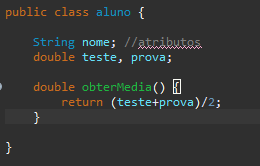
A recursividade é um conceito poderoso em programação que permite que um método chame a si mesmo dentro de sua própria definição. É como se olhássemos para um espelho que reflete outro espelho, criando uma ilusão de profundidade infinita. A recursividade é útil para resolver problemas que podem ser divididos em subproblemas menores do mesmo tipo. A sobrecarga de métodos permite que tenhamos múltiplos métodos com o mesmo nome, mas com diferentes parâmetros. É como se tivéssemos várias chaves de fenda com diferentes tamanhos para diferentes tipos de parafusos.

A sobrecarga facilita a escrita de código mais limpo e organizado. A passagem de parâmetros por referência, utilizada em Java, significa que, quando passamos um objeto como parâmetro para um método, estamos passando uma referência para o objeto original, e não uma cópia. Qualquer alteração feita no objeto dentro do método afetará o objeto original.

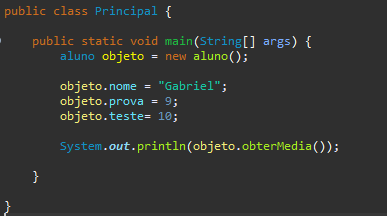
Por fim, a anotação "@Override" é utilizada para indicar que um método está sendo sobrescrito de uma superclasse. Isso garante que estamos realmente sobrescrevendo o método e não criando um novo método por engano.

**Classe:**

A planta do objeto!



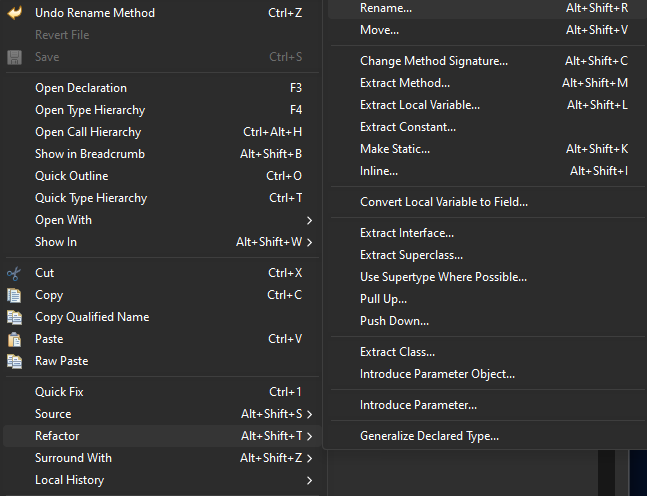
Como posso fazer com que outra classe chame essa classe aluno?



**Objeto:**

Materialização da classe, utiliza new pois não é um tipo primitivo.

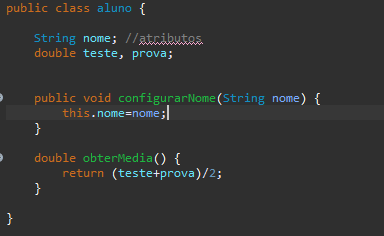
DICA:

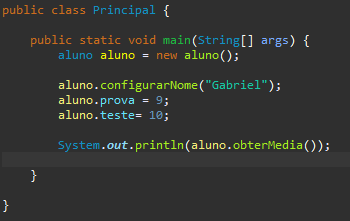


Usando refactor ele muda o nome em todos os lugares! (da palavra selecionada)

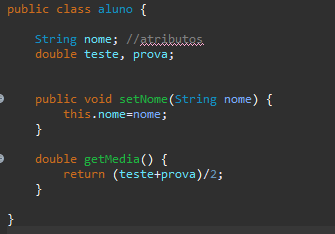
**Encapsulamento:**

É recomendado que toda manipulação dos dados não pode ser dentro da classe! Mas fora. Assim:

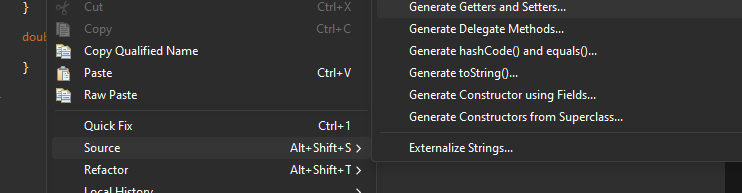




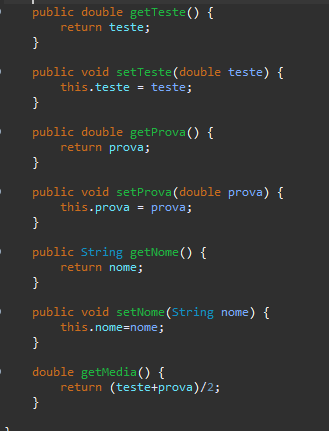
Só que o pessoal não faz assim! tem uma boa pratica para isso.(na nomenclatura):



Quando quiser ter acesso para consultar ou alterar usar a palavra: get ou set



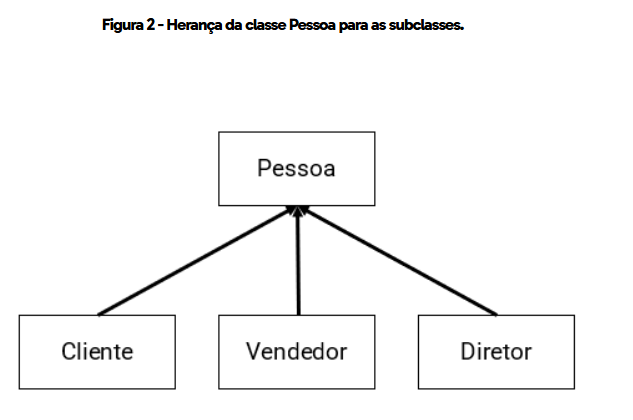
Ele faz automaticamente de todas as variaveis:



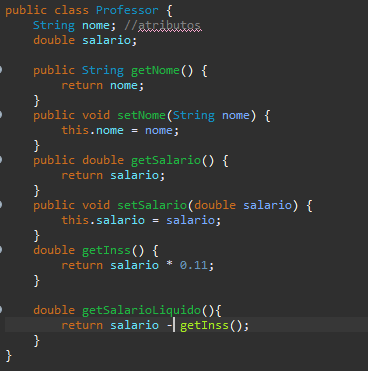
**Herança e classe abstrata:**

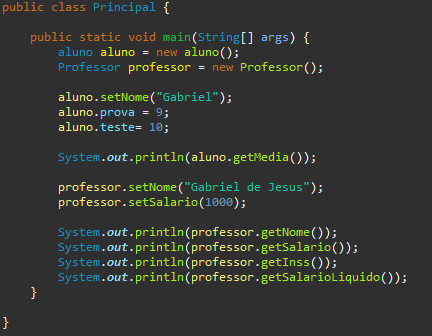
Para que repetir código? Bora aproveitar código já escrito.

herança é um mecanismo disponível nas linguagens de programação que se apoiam no paradigma de objetos que possibilita que características comuns a diversas classes sejam reunidas em uma classe base. A partir desta base, as demais classes podem ser especificadas de modo que cada classe derivada (ou subclasse) apresente características e comportamentos herdados da classe base e outras que foram definidas exclusivamente para elas. Na prática, isso significa reunir todas as características comuns entre várias entidades e a partir disso criar uma base em que todas elas se baseiam. Em resumo, as características e comportamentos de uma classe (base) são herdadas a outras classes (subclasses).



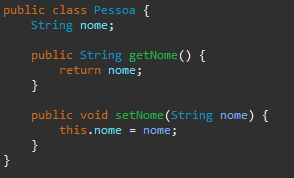
Herança é sempre utilizada em Java, mesmo que não explicitamente. Quando uma classe é criada e não há nenhuma referência à sua classe base, implicitamente a classe criada é derivada diretamente da classe Object. É por esse motivo que todos os objetos podem invocar os métodos da classe Object, tais como equals() e toString().



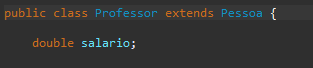
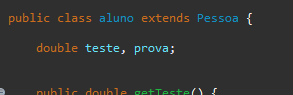


A herança é para que o que é em comum entre a classe, eu não precise escrever 2 vezes! Mas como posso fazer isso?

Nos 2 (aluno e professor) não tem o nome? São pessoas no caso? Então:



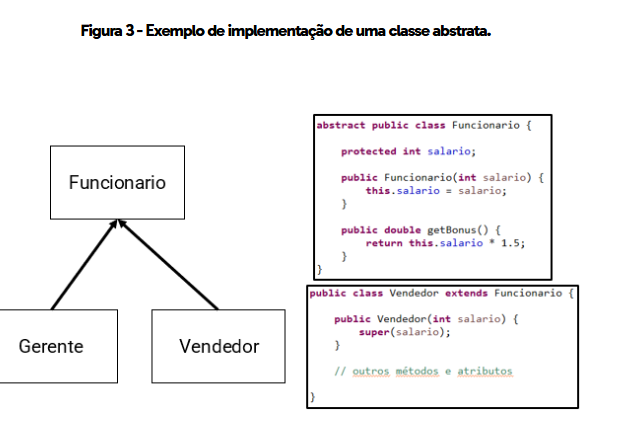
E como passa isso para os alunos e professores? É colocando na classe do aluno/professor o nome extends Pessoa.



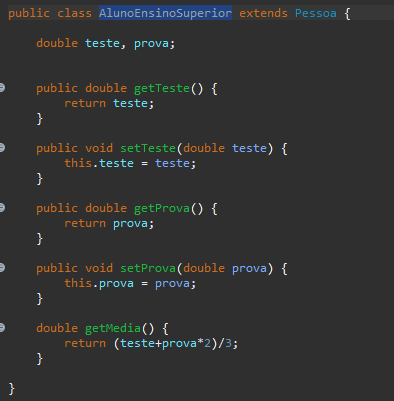
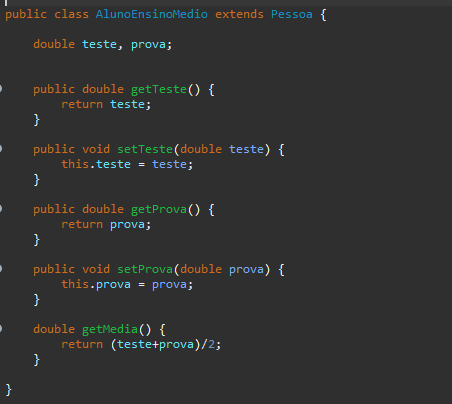
A herança tem que ser para o mesmo tipo: ex: pessoa = aluno e professor. Se fosse cachorro não daria certo por exemplo.

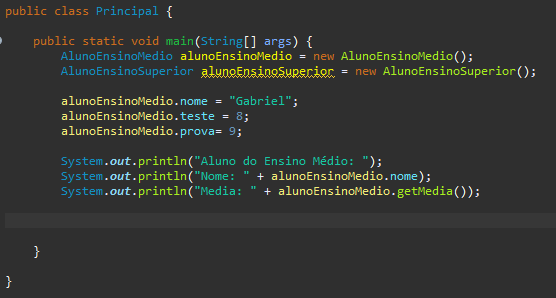
Já a **Classe abstrata** tem métodos que obriga que suas filhas implementarem esses métodos.

As abstratas são classes que estruturam um tipo mas que não podem ser instanciadas. É quase como se fosse uma interface, a diferença é que elas podem implementar comportamentos e características, mas nunca serão usadas diretamente, somente as classes que a herdarem poderão usá-la.



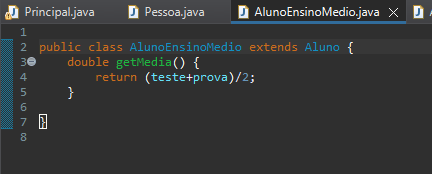
Vamos colocar aluno ensino médio e aluno ensino superior (esse com a prova com peso 2).

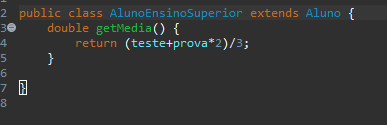


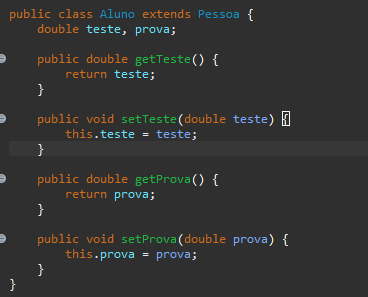


E agora se eu quiser copiar para o ensino superior?

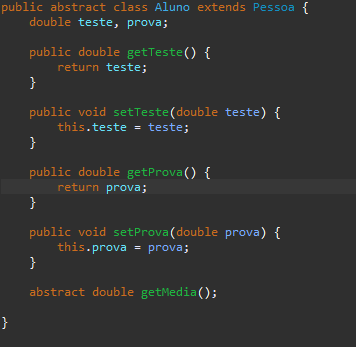
É melhor criar uma classe aluno e o que tiver igual jogar la:







Mas e esse getmedia? Como posso fazer para que ele vá para classe aluno?



Fazendo isso, todos os filhos tem que ter o getMedia();

**Interface:**

Uma classe totalmente abstrata, todos os métodos ser abstrato.

A sua função é fazer com que todas as classes associadas a elas sejam “obrigadas” a cumprir um "acordo’'. Este acordo será possuir as características e comportamentos definidos por ela. Se ela diz que todos devem ter um método pagar(), por exemplo, todas as classes associadas a esta interface terão que, obrigatoriamente, implementar este método.