Sala de aula invertida

Definição de POO

Programação orientada a objetos (POO, ou OOP segundo as suas siglas em inglês) é um paradigma de programação baseado no conceito de "objetos", que podem conter dados na forma de campos, também conhecidos como atributos, e códigos, na forma de procedimentos, também conhecidos como métodos.

Características da programação OO

A POO trabalha utilizando classes, ela é quem dá a vida ao objeto, pois nela é que são criadas as funcionalidades através dos métodos, dizendo se o objeto vai andar parar rodar, etc., e suas especificidades através dos atributos, sua cor, seu nome, etc.. a classe não é o objeto, ela é utilizada para construí-lo.

O objeto é uma abstração de software que pode ser algo representado por algo real ou virtual, que é formado por um conjunto de propriedades (variáveis) e procedimentos (métodos), onde as variáveis possuem um tipo que define os possíveis valores que pode representar um número inteiro, real ou *string* e, os métodos são rotinas que, quando executadas realizam algumas tarefas como alterar o conteúdo de uma variável do objeto.A classe é a abstração do objeto. Na realidade ao programarmos um objeto, definindo suas características e funcionalidades, estamos programando uma classe.

Dentre alguns recursos aplicados na programação orientada a objeto, temos: encapsulamento, herança e polimorfismo.

Encapsulamento é a limitação imposta a atributos de uma classe, restringindo o acesso aos métodos dessa classe através de limitadores de acesso, dos quais os mais utilizados são:

Público (*public*) – a classe só é vista pelas demais classes se estiverem dentro do mesmo pacote;

Protegida (*protected*) – o acesso se restringe apenas a própria classe e suas subclasses;

Privado (*private*) – o acesso só é permitido somente pela própria classe.

O conceito de herança nos permite definir uma nova classe, com base em uma já existente. A classe criada (subclasse ou classe derivada) automaticamente herda todas as variáveis e métodos da classe já existente (superclasse). O mecanismo de herança permite que a subclasse inclua ou sobre ponha novas variáveis e métodos da superclasse. O mecanismo de herança é recursivo, permitindo criar-se uma hierarquia de classes, do nível mais alto, onde as características herdadas são comuns a todos os objetos desta classe até os níveis inferiores, onde estão as especializações das classes superiores. As subclasses herdam as características comuns, definindo suas propriedades específicas, ou seja, facilita a construção de novos objetos a partir de outros já existentes, sem a necessidade de reescrevê-lo todo.

Polimorfismo: A palavra polimorfismo utilizada na biologia para definir várias formas de membros de uma mesma espécie. Utilizando essa analogia, o mecanismo de polimorfismo utilizado na POO, nos permite tratar objetos semelhantes de uma maneira uniforme.

Para que o polimorfismo seja utilizado, é necessária a aplicação do conceito de herança, aplicados somente aos métodos da classe.

Anterior a utilização da POO, sempre que uma nova funcionalidade fosse criada ou acrescentada, a aplicação deveria ser alterada e recompilada, porém com a utilização do polimorfismo, acrescentar novos métodos a classe já existente sem a necessidade de ter que recompilar a aplicação tornou-se mais fácil.

Com base nos conceitos de objetos, classes, encapsulamento, herança e polimorfismo, que são considerados como o paradigma da POO representando uma forma inovadora de pensar e criar novos softwares, trazendo inúmeros benefícios e valores agregados no desenvolvimento deles, tais como: a reutilização de códigos que reduz o tempo e o risco no desenvolvimento, aumentando assim a confiabilidade do software, haja vista que, os códigos reutilizados já foram testados, facilita também a compreensão da linguagem com desenvolvedor na hora da manutenção de softwares complexos.

Os Pilares da POO

### Abstração

Essa característica é muito importante para definir o paradigma da programação orientada a objetos. A abstração consiste em transformar e extrair informações do mundo real para dentro do código.

Imagine que, em nossa aplicação, possuímos uma classe chamada Carro. Essa classe terá características padrões a serem seguidas, como, por exemplo, a marca, a quantidade de portas, a cor, o ano, entre outras. Repare que estamos utilizando informações reais sobre veículos.

Essa classe também pode possuir métodos e funcionalidades únicas, como, por exemplo, um método de ligar o carro, acender o farol ou, até mesmo, abrir o porta-mala.

### Encapsulamento

Lembra quando falamos que um objeto na programação é um pequeno bloco contendo dados e informações? Isso só é possível devido ao encapsulamento.

Dessa forma, garantimos que aquelas informações serão de responsabilidade exclusiva daquele próprio objeto, definindo um escopo para a sua atuação.

### Herança

Esse conceito se assemelha muito ao que conhecemos como herança na vida real. Consiste em passar informações e características de uma classe para outra, sem a necessidade de repetição de código e entidades.

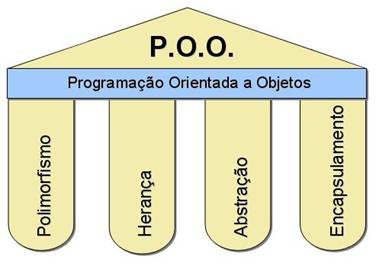
Usando novamente o exemplo do carro, caso agora quiséssemos criar uma classe Moto, poderíamos herdar algumas propriedades comuns em veículos, como, por exemplo, a marca, as rodas e a funcionalidade de ligar/desligar.

### Polimorfismo

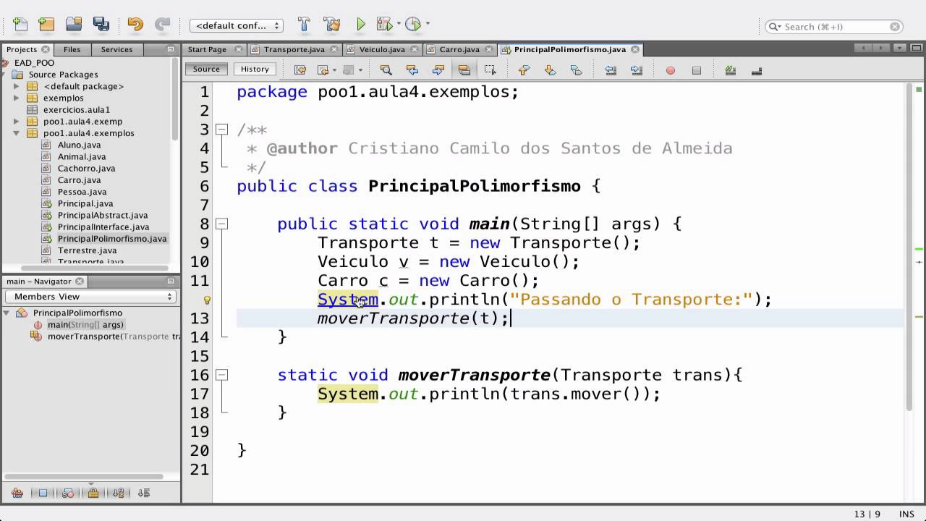
Polimorfismo significa “muitas formas”. Na programação orientada a objetos, essa característica define a especialização das classes filhas.

Voltando ao exemplo da classe Moto. Lembra que herdamos algumas características em comum de veículos da classe Carro? Pois é, acontece que, diferente dos carros, as motos possuem apenas 2 rodas. Cada veículo também possui uma “identidade” própria e uma marca diferente.

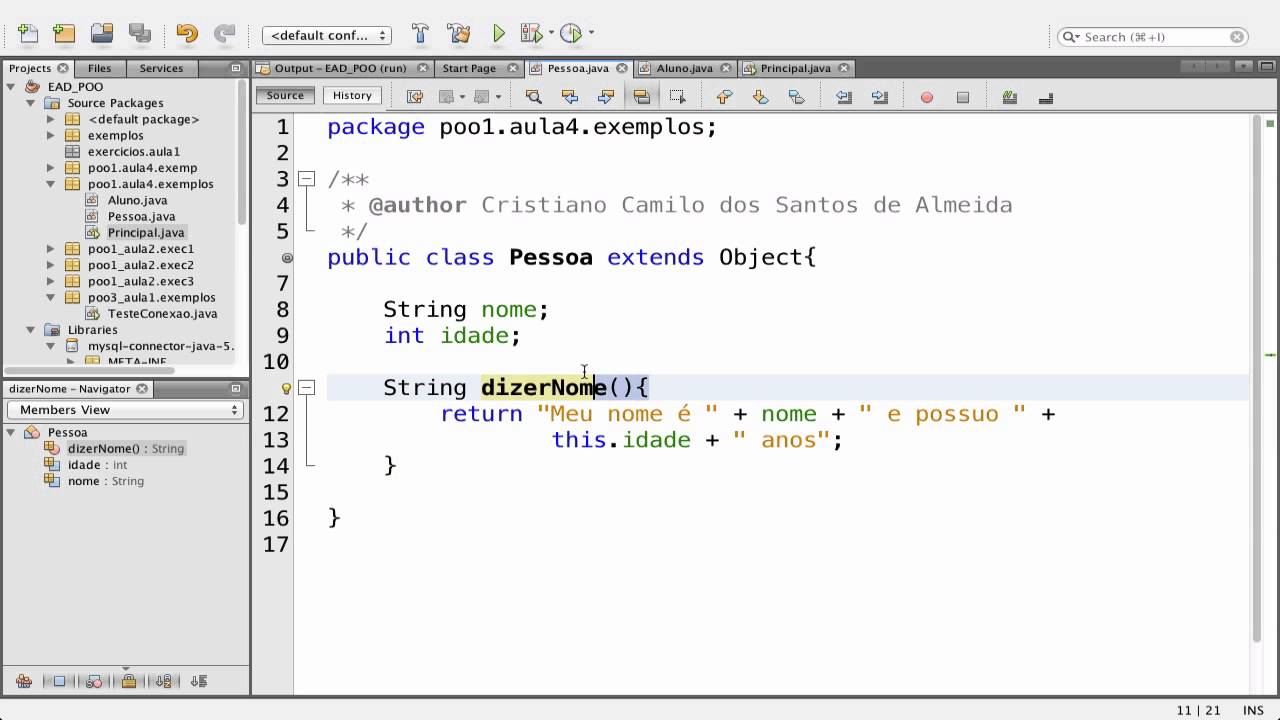
Percebe que apesar de serem propriedades e características em comum, os seus valores diferem? É exatamente essas possibilidades que o polimorfismo traz para os objetos na programação.



exemplos em JAVA de polimorfismo



exemplos em JAVA de herança



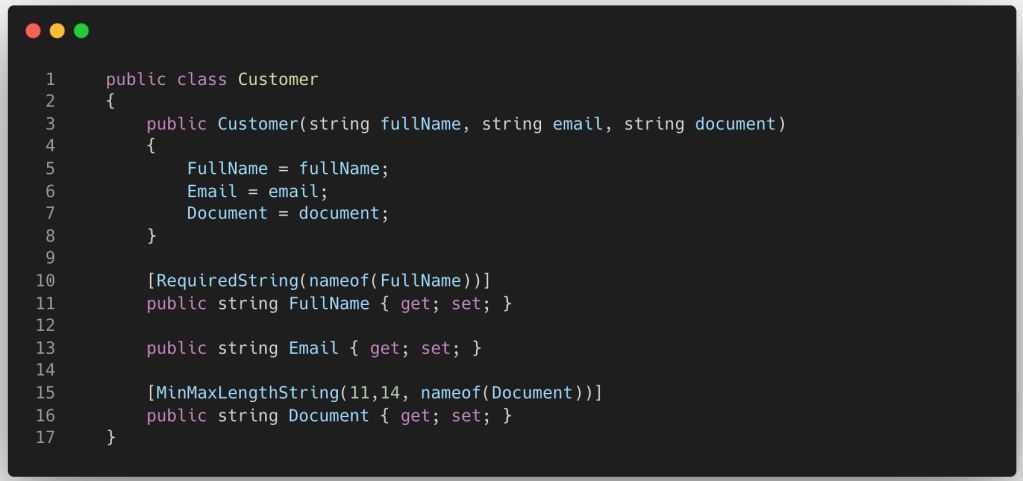
O que são atributos ?

Os **Atributos** em [Programação Orientada a Objetos](https://pt.wikipedia.org/wiki/Orienta%C3%A7%C3%A3o_a_objeto) são os elementos que definem a estrutura de uma [classe](https://pt.wikipedia.org/wiki/Classe_(programa%C3%A7%C3%A3o)). Os atributos também são conhecidos como [variáveis](https://pt.wikipedia.org/wiki/Vari%C3%A1veis) de classe, e podem ser divididos em dois tipos básicos: atributos de [instância](https://pt.wikipedia.org/wiki/Inst%C3%A2ncia_(classe)) e de classe.

Os valores dos atributos de instância determinam o estado de cada classe. Um atributo de classe possui um estado que é compartilhado por todos os objetos de uma classe. Atributos de classe podem ser chamados também de atributos estáticos ou dinâmicos. Constantes são considerados atributos de classe quando estão fora de qualquer método.[[1]](https://pt.wikipedia.org/wiki/Atributo_(computa%C3%A7%C3%A3o)#cite_note-Attributes-1)

As mensagens enviadas a um objeto (chamada de um [método](https://pt.wikipedia.org/wiki/M%C3%A9todo_(programa%C3%A7%C3%A3o))) podem mudar o valor de um ou mais atributos, alterando o estado de um objeto.

Um atributo é um dado para o qual cada objecto tem seu próprio valor. Atributos são, basicamente, a estrutura de dados que vai representar a classe.

[](atributo)

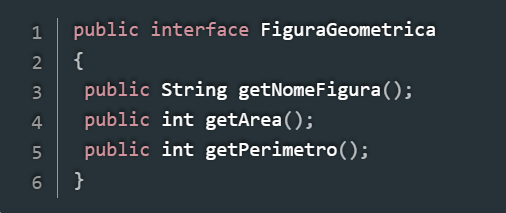
O que é uma interface em Java?

A interface é um recurso muito utilizado em Java, bem como na maioria das linguagens orientadas a objeto, para “obrigar” a um determinado grupo de classes a ter métodos ou propriedades em comum para existir em um determinado contexto, contudo os métodos podem ser implementados em cada classe de uma maneira diferente. Pode-se dizer, a grosso modo, que uma interface é um contrato que quando assumido por uma classe deve ser implementado.

### Utilizando interface em Java

Dentro das interfaces existem somente assinaturas de métodos e propriedades, cabendo à classe que a utilizará realizar a implementação das assinaturas, dando comportamentos práticos aos métodos.

Abaixo é possível ver um exemplo de uma interface chamada FiguraGeometrica com três assinaturas de métodos que virão a ser implementados pelas classes referentes às figuras geométricas.



O que são enumerações?

### Enum no Java

São tipos de campos que consistem em um conjunto fixo de constantes (static final), sendo como uma lista de valores pré-definidos. [**Na linguagem de programação Java**](https://www.devmedia.com.br/cursos/java), pode ser definido um tipo de enumeração usando a palavra chave **enum**.

Todos os tipos enums implicitamente estendem a classe java.lang.Enum, sendo que o Java não suporta herança múltipla, não podendo estender nenhuma outra classe.

### Características dos tipos enum

Em relação às propriedades é preciso tomar os seguintes cuidados:

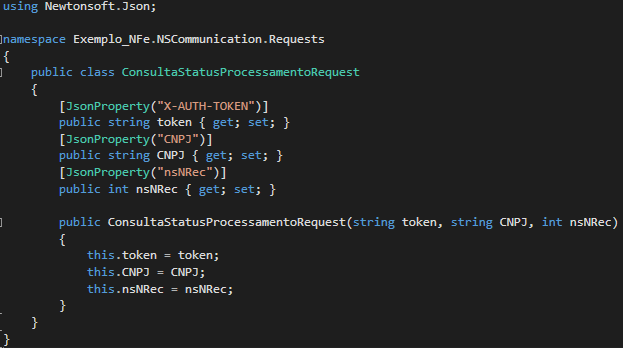
* As instâncias dos tipos enum são criadas e nomeadas junto com a declaração da classe, sendo fixas e imutáveis (o valor é fixo).;
* Não é permitido criar novas instâncias com a palavra chave new;
* O construtor é declarado private, embora não precise de modificador private explícito;
* Seguindo a convenção, por serem objetos constantes e imutáveis (static final), os nomes declarados recebem todas as letras em MAIÚSCULAS;
* As instâncias dos tipos enum devem obrigatoriamente ter apenas um nome;
* Opcionalmente, a declaração da classe pode incluir variáveis de instância, construtor, métodos de instância, de classe, etc.

O que é Serialização de Objetos em Java?

A serialização é quando um objeto é transformado, em uma cadeia de bytes e desta forma pode ser manipulado de maneira mais fácil, seja através de transporte pela rede ou salvo no disco. Após a transmissão ou o armazenamento esta cadeia de bytes pode ser transformada novamente no objeto Java que o originou. Em suma, a ideia por trás da serialização é a de "congelar" o objeto, guardando-o por um tempo indeterminado, movê-lo e depois "descongelar" esse objeto tornando novamente utilizável. Cenários comuns para o uso da serialização de objetos dentro do Java, são as invocações remotas de métodos através de objetos distribuídos, aplicações que fazem uso do mapeamento objeto/relacional e servidores de aplicações quando um cliente está desatualizado em relação a versão dos jars necessários.

A serialização trabalha apenas com atributos de instância de uma classe, não incluindo os atributos estáticos. Outro detalhe importante da serialização é que se o objeto a ser serializado for proveniente de uma subclasse, todos os atributos de instancia, mesmos aqueles originários de superclasses, serão serializados. Também se um objeto contiver referências de outros objetos, todas as referências serão serializadas.

Para um objeto estar credenciado a passar pelo processo de serialização sua classe deve implementar a interface **java.io.Serializable** que sinalizará a máquina virtual Java (JVM) que objetos daquela classe estão aptos a serem serializadas. Caso não se deseje serializar algum atributo de instância específico de um determinado objeto, basta sinalizá-lo como **transient**, assim o objeto serializado não conterá a informação referente a este atributo.



O que é relacionamento de objetos?

O que é relacionamento de objetos?

O relacionamento entre objetos define como eles vão interagir ou colaborar para executar uma operação em uma aplicação. Em qualquer aplicação, objetos de classes de interface do usuário vão interagir com objetos da camada de negócios para executar uma operação. Além disso, os objetos da camada de negócios podem interagir com objetos de um repositório que por sua vez se comunica com algum objeto da fonte de dados ou com algum objeto de serviço.

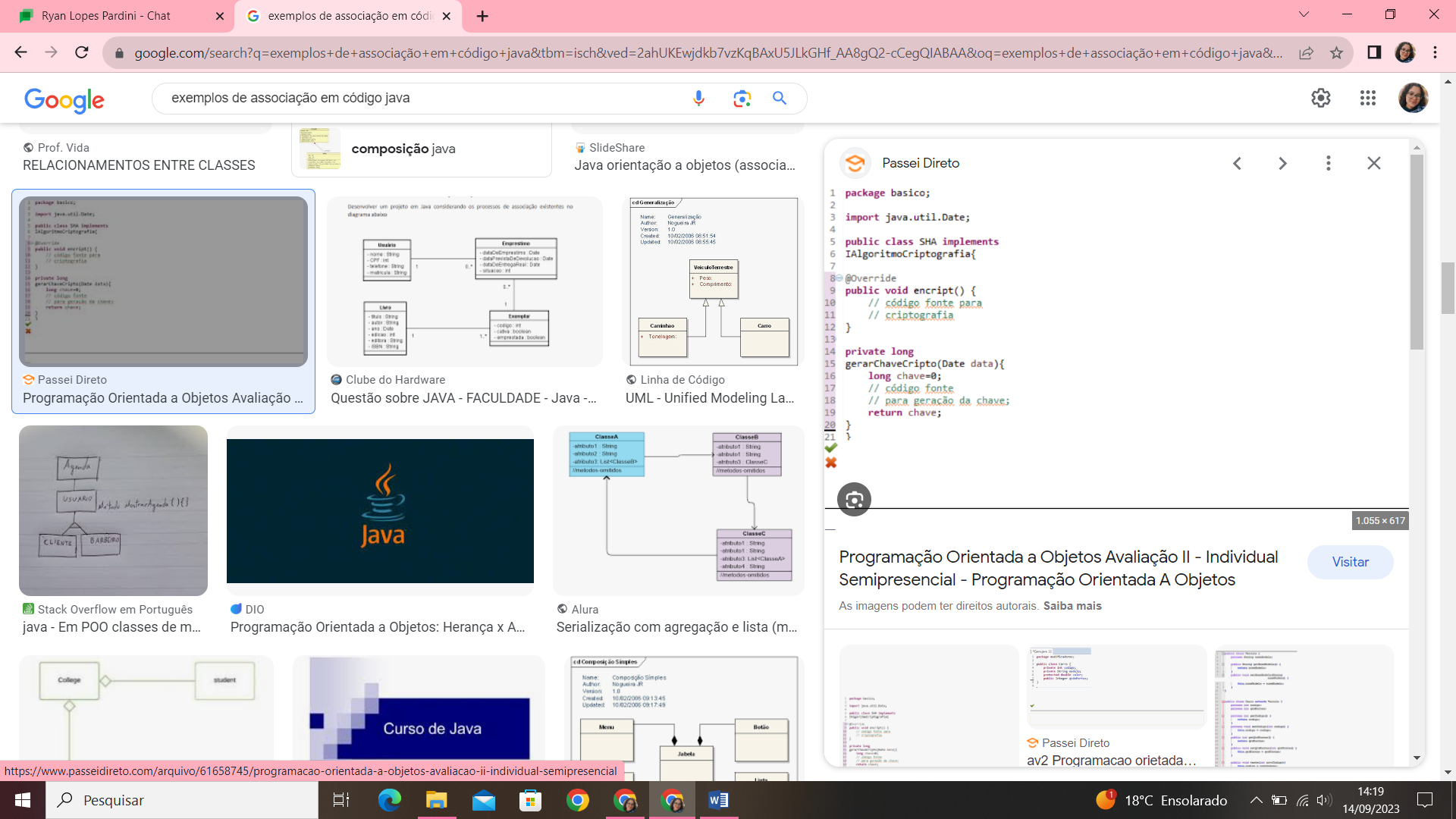
Os tipos básicos de relacionamentos definidos na programação orientada a objetos são:

1. Associação
2. Colaboração
3. Agregação
4. Composição
5. Herança

**1 - Associação**

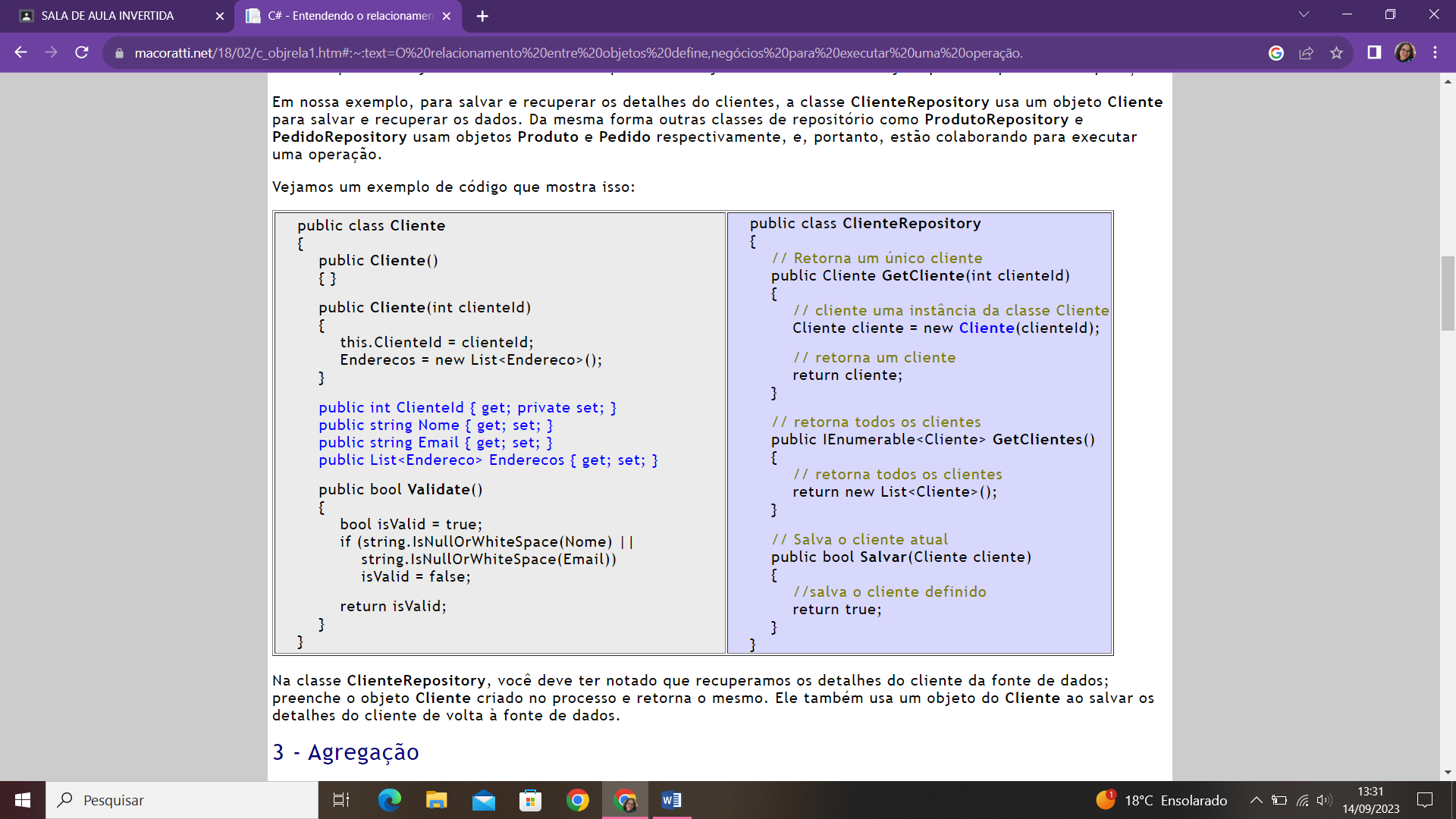
A associação descreve um vínculo que ocorre entre classes, sendo que a mais comum é a  associação binária, mas é possível que uma classe esteja vinculada a si própria, e ai temos a associação unária; outro cenário é onde temos uma associação que seja compartilhada por mais de uma classe, o que conhecemos por associação ternária ou N-ária, sendo este tipo de associação a mais rara e também mais complexa.

Falamos sobre associação entre dois objetos quando cada um deles pode usar o outro, mas também cada um deles pode existir sem o outro. Não há dependência entre eles.



**2 - Colaboração**

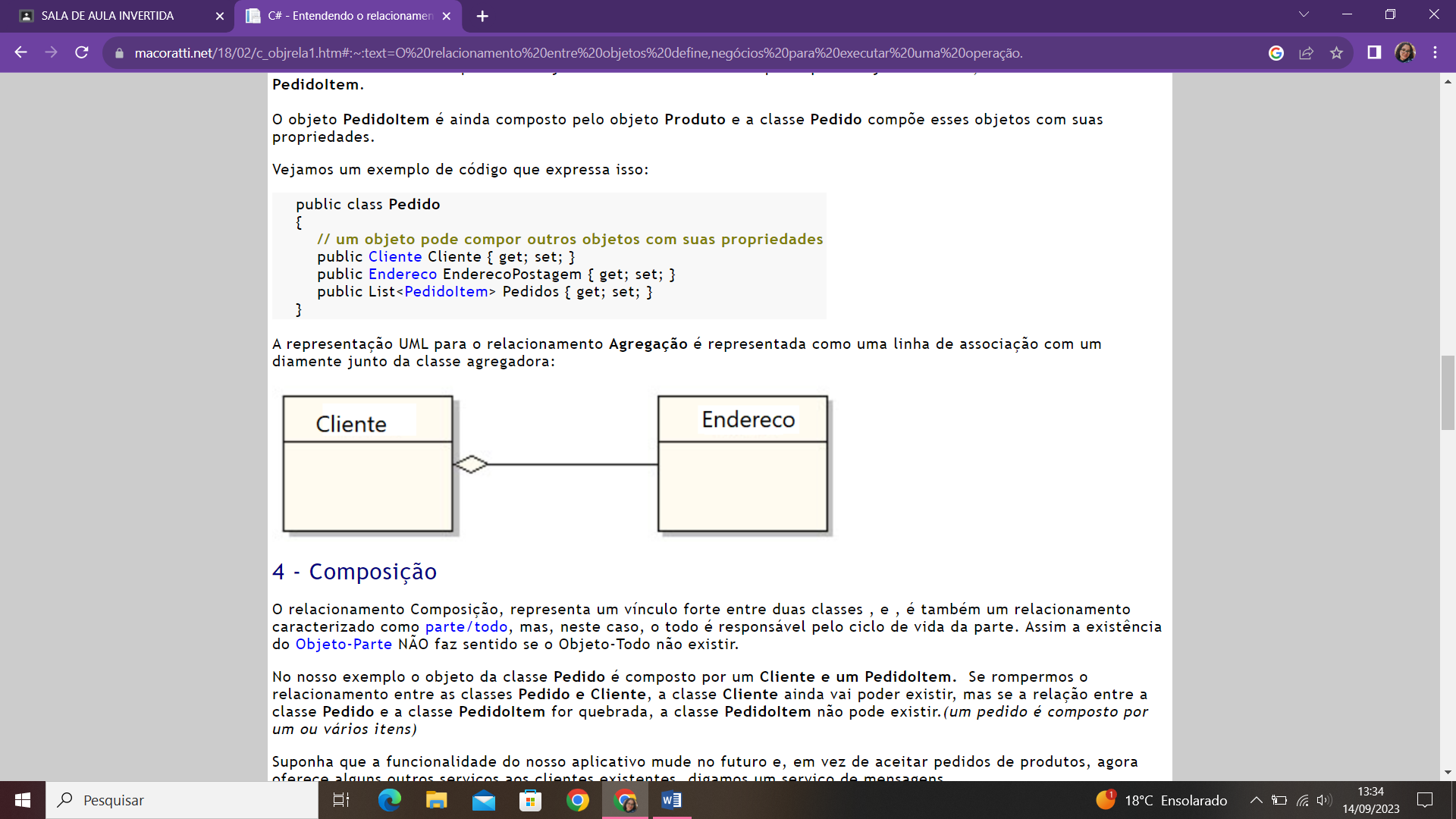
O relacionamento de colaboração é muitas vezes conhecido como relacionamento **'usa um' ou 'uses a'**.  A colaboração declara que dois objetos estão colaborando quando um objeto faz uso de outro objeto para completar uma operação.  
  
Nesse exemplo, para salvar e recuperar os detalhes do cliente, a classe **ClienteRepository** usa um objeto **Cliente** para salvar e recuperar os dados. Da mesma forma outras classes de repositório como **ProdutoRepository** e **PedidoRepository** usam objetos **Produto** e **Pedido** respectivamente, e, portanto, estão colaborando para executar uma operação.



**3 - Agregação**

Esse relacionamento é um tipo especial de associação onde as informações de um objeto *(chamado objeto-todo)*precisam ser complementados pelas informações contidas em um ou mais objetos de outra classe *(chamados objetos-parte)*; temos então o que conhecemos como **todo/parte**.  A agregação um tipo composição que representa um vínculo fraco entre duas classes.

O relacionamento de agregação as vezes é referido como relacionamento "**tem um" ou 'has a'.**  
  
Neste tipo de relacionamento, um objeto pode ser composto de um ou mais objetos na forma de suas propriedades. Assim temos que :  
  
- Todo **Cliente** tem um endereço para o qual o produto solicitado será enviado  
- Cada **Pedido** tem um cliente, um endereço de envio e um produto representado como um **PedidoItem**  
  
Podemos então concluir que nosso objeto da classe **Pedido** é composto pelos objetos **Cliente**, **Endereco** e **PedidoItem**.  
  
O objeto **PedidoItem** é ainda composto pelo objeto **Produto** e a classe **Pedido** compõe esses objetos com suas propriedades.





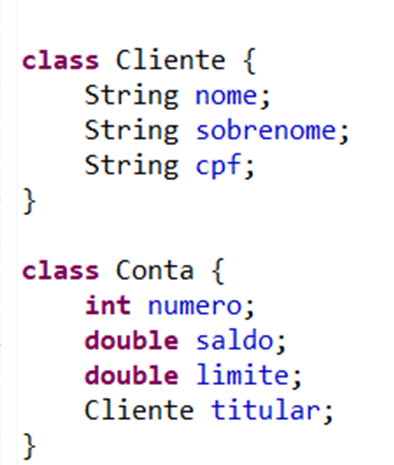
**4 - Composição**

O relacionamento Composição, representa um vínculo forte entre duas classes, e é também um relacionamento caracterizado como parte/todo, mas, neste caso, o todo é responsável pelo ciclo de vida da parte. Assim a existência do Objeto-Parte NÃO faz sentido se o Objeto-Todo não existir.

No nosso exemplo o objeto da classe **Pedido** é composto por um **Cliente e um PedidoItem.** Se rompermos o relacionamento entre as classes **Pedido e Cliente**, a classe **Cliente** ainda vai poder existir, mas se a relação entre a classe **Pedido** e a classe **PedidoItem** for quebrada, a classe **PedidoItem** não pode existir. Um pedido é composto por um ou vários itens.

Suponha que a funcionalidade do nosso aplicativo mude no futuro e, em vez de aceitar pedidos de produtos, agora oferece alguns outros serviços aos clientes existentes, digamos um serviço de mensagens.

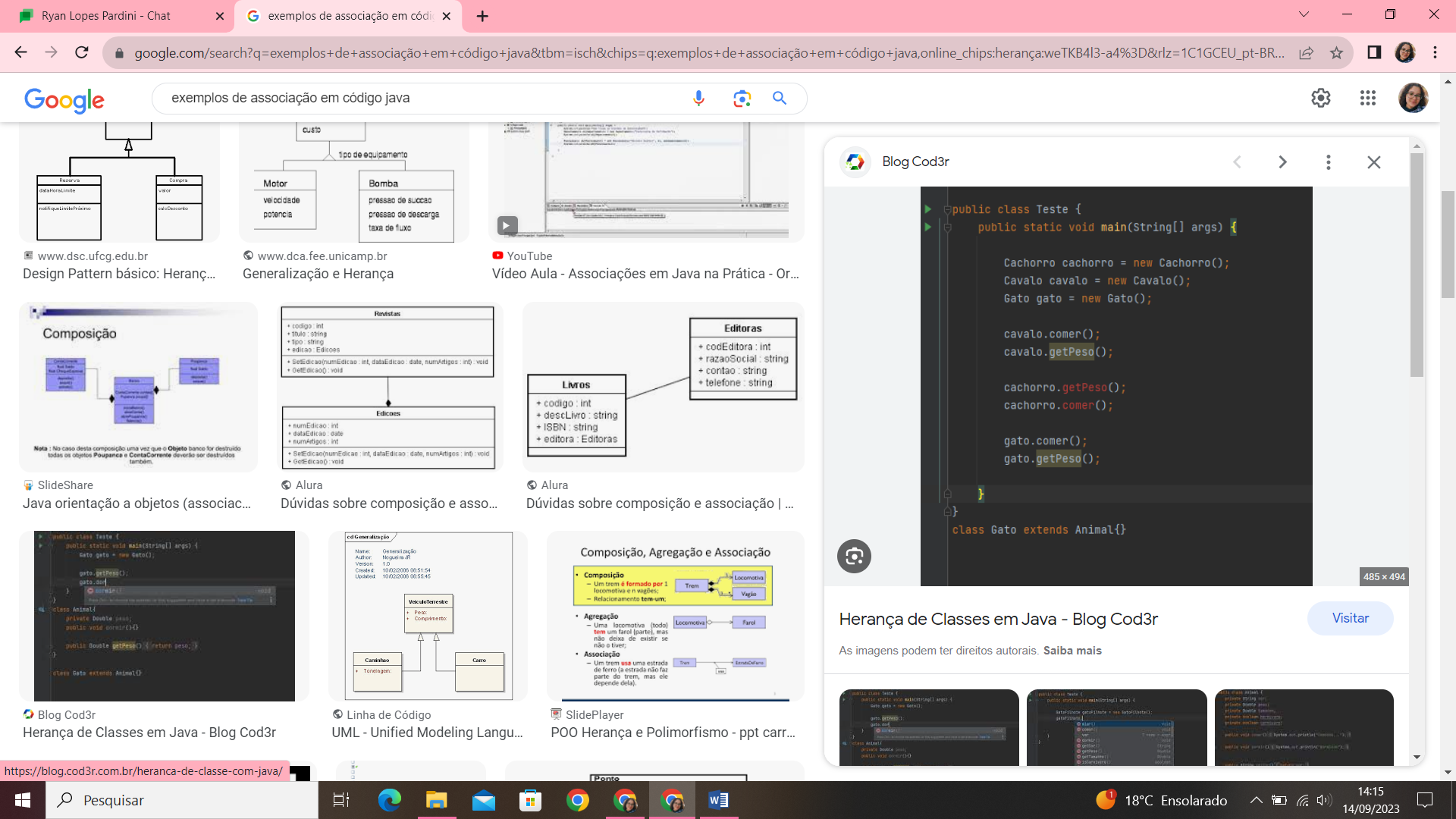
Nesse cenário, a classe **Pedido** não servirá para nada. No entanto, a classe **Cliente** que já foi composta pela classe **Pedido** ainda pode existir sem ela, já a classe **PedidoItem** não pode.



**5 - Herança**

A herança às vezes é referida como um relacionamento**"é um" ou 'is a'**. Neste tipo de relacionamento, uma classe herda os membros de outra classe. A classe herdada é conhecida como a classe base, enquanto a classe herdada é conhecida como a classe derivada. Como a classe derivada tem os membros da classe base, pode-se dizer que a classe derivada é um subtipo da classe base. A classe derivada pode ou não ter membros diferentes dos herdados.  
  
Suponha que nosso aplicativo esteja funcionando bem no mercado. Ao ver isso, o proprietário do produto agora quer adicionar um novo recurso no aplicativo que monitoraria o tipo de produtos com alta demanda.

A partir do novo requisito, é bastante claro que teremos de criar subtipos de nossa classe de produtos. Esses subtipos representarão as categorias de produtos especializados no mundo real.



As classes **AlbumMusica e Livro**, possuem algumas propriedades próprias; mas como herdam da classe **Produto**, elas também herdam suas propriedades.

Então, pode-se dizer que o **AlbumMusica** é um tipo de produto e, da mesma forma o **Livro** é um tipo de produto.

Dessa forma a herança pode ser usada para reutilizar e estender código.

