# Modificador *final*

Fazendo uma revisão sobre *override*, que é quando mudamos o comportamento do método na sua própria classe, definido de forma diferente que foi definido na sua superclasse. Agora iremos falar sobre um modificador que afeta diretamente à herança, que é o modificador final.

Este modificador tem várias utilidades dependendo da onde usa. Podemos usá-lo em métodos, classes e/ou atributos.

## A) Atributos:

Primeiro criemos uma classe chamada carro:

|  |
| --- |
| **public** **class** Carro  { **private** **double** velocidadeLimite;  **private** String nome;  **private** String marca;    **public** String getNome() {**return** nome;}  **public** **void** setNome(String nome) {**this**.nome = nome;}    **public** String getMarca() {**return** marca;}  **public** **void** setMarca(String marca) {**this**.marca = marca;}    @Override **public** String toString()  {**return** "Carro [nome=" + nome + ", marca=" + marca + "]";}  } |

Digamos que nosso carro possui um limite de velocidade, e este parâmetro não pode mudar, independente do que aconteça. São conhecidas como constantes, e no java as definimos com o fator *static*. Como sabemos que nossa *double* velocidade limite, por legislação, não deve variar independente da instância da classe carro, então simplesmente colocamos este fator antes do tipo da variável declarada e, além disso, devemos já inicializa-la com um valor.

|  |
| --- |
| **public** **class** Carro  { **private** **static** **double** velocidadeLimite;  .  .  .  .  } |

Mas ainda assim, não temos uma forma de impedir que não mude. Então, como queremos que nosso atributo seja constante, independente da instância da classe, devemos complementar nosso atributo estático com o fator ***final* e ainda inicializá-lo com um valor**.

|  |
| --- |
| **public** **class** Carro  { **private** **static** **final** **double** ***velocidadeLimite*** = 130;  **private** String nome;  **private** String marca;    **public** String getNome() {**return** nome;}  **public** **void** setNome(String nome) {**this**.nome = nome;}    **public** String getMarca() {**return** marca;} prá  **public** **void** setMarca(String marca) {**this**.marca = marca;}    **public** **static** **double** getVelocidadeLimite(){**return** ***velocidadeLimite***;}  **public** **static** **void** setVelocidadeLimite(**double** velocidadeLimite)  {Carro.***velocidadeLimite*** = velocidadeLimite;}    @Override **public** String toString()  {**return** "Carro [nome=" + nome + ", marca=" + marca + "]";}  } |

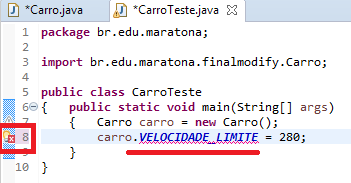
Uma boa prática é sempre nomear essas variáveis com letras maiúsculas, e se tiver que separar palavras, usar “­\_”.

|  |
| --- |
| **public** **class** Carro  { **public** **static** **final** **double** ***VELOCIDADE\_LIMITE*** = 130;  **private** String nome;  **private** String marca;    **public** String getNome() {**return** nome;}  **public** **void** setNome(String nome) {**this**.nome = nome;}    **public** String getMarca() {**return** marca;}  **public** **void** setMarca(String marca) {**this**.marca = marca;}    @Override **public** String toString()  {**return** "Carro [nome=" + nome + ", marca=" + marca + "]";}  } |

**OBS: Caso não declararmos nosso atributo constante com fator *static*, podemos inicializa-la através de um construtor. Ou, quando não declaramos como *final*, podemos fazer por bloco de inicialização.**

|  |
| --- |
| **public** **class** Carro  { **public** **final** **double** VELOCIDADE\_LIMITE;  **private** String nome;  **private** String marca;    **public** Carro() {**this**.VELOCIDADE\_LIMITE = 130;}    **public** String getNome() {**return** nome;}  **public** **void** setNome(String nome) {**this**.nome = nome;}    **public** String getMarca() {**return** marca;}  **public** **void** setMarca(String marca) {**this**.marca = marca;}    @Override **public** String toString()  {**return** "Carro [nome=" + nome + ", marca=" + marca + "]";}  } |
| **public** **class** Carro  { //private static final double VELOCIDADE\_LIMITE = 130;  **public** **static** **double** *VELOCIDADE\_LIMITE*;  **private** String nome;  **private** String marca;    **static** {*VELOCIDADE\_LIMITE* = 130;}    **public** String getNome() {**return** nome;}  **public** **void** setNome(String nome) {**this**.nome = nome;}    **public** String getMarca() {**return** marca;}  **public** **void** setMarca(String marca) {**this**.marca = marca;}    @Override **public** String toString()  {**return** "Carro [nome=" + nome + ", marca=" + marca + "]";}  } |

Vamos utilizar o primeiro bloco de código e testar em uma nova classe de teste chamada CarroTeste.java com método *main*. Se tentarmos definir uma velocidade limite para nossa instância de carro, perceba que teremos já um erro de compilação, pois o java já irá entender que estamos tentando alterar algo que não deve.

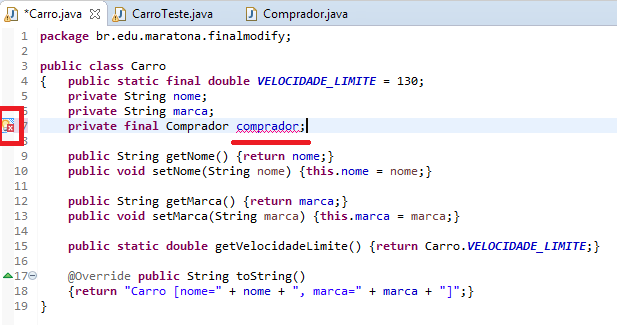


**Figura – Erro de compilação. Alteração de uma constante.**

Mas isso quando temos uma variável do tipo primitivo, agora o que será que acontece quando temos uma variável do tipo Referência, ou seja, uma classe. Criemos uma nova classe do tipo Comprador.java.

|  |
| --- |
| **public** **class** Comprador  { **private** String nome;    **public** String getNome() {**return** nome;}  **public** **void** setNome(String nome) {**this**.nome = nome;}  } |

Se declararmo que essa classe é um atributo da classe Carro, como *final*, veja que teremos um erro novamente de compilação.

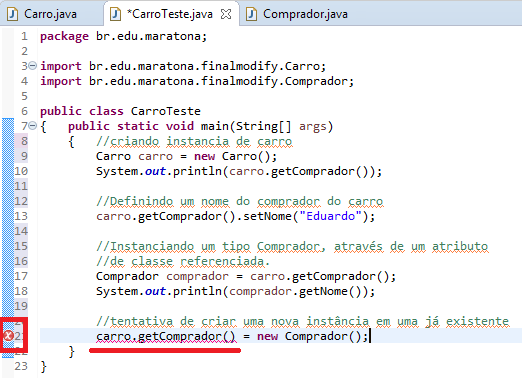


**Figura – Erro de compilação. Declaração de classe, como atributo, tipo final.**

Se quisermos corrigir isso, já vimos em momentos anteriores que basta inicializa-lo, por exemplo:

|  |
| --- |
| **public** **class** Carro  { **public** **static** **final** **double** ***VELOCIDADE\_LIMITE*** = 130;  **private** String nome;  **private** String marca;  **private** **final** Comprador comprador = **new** Comprador();    **public** String getNome() {**return** nome;}  **public** **void** setNome(String nome) {**this**.nome = nome;}    **public** String getMarca() {**return** marca;}  **public** **void** setMarca(String marca) {**this**.marca = marca;}    @Override **public** String toString()  {**return** "Carro [nome=" + nome + ", marca=" + marca + "]";}  } |

**Obs: outro erro comum é criar uma nova referência em algo que já está referenciado.**

****

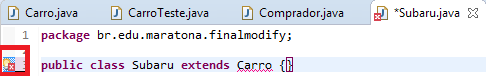
**Figura – Erro de compilação. Atribuindo uma nova instância em um atributo já referenciado e como *final*.**

## B) Classes

Agora iremos definir que nossa classe Carro é do tipo *final*. Isso implica que essa classe não pode ser extendida, ou seja, não poderá ter classes filhas pois é a última das classes.

|  |
| --- |
| **public** **final** **class** Carro  {.  .//bloco de códigos  .  } |

Se quisermos tentar criar uma classe que herda dessa classe final, teremos mais um erro de compilação, pois uma classe definida como final, não podemos ter classes que herde da mesma.

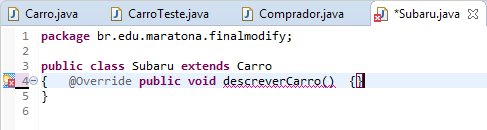


**Figura – Erro de compilação. Classe herdando de uma classe do tipo *final*.**

## C) Métodos

Para fecharmos esse assunto, podemos usar esse fator *final* em métodos também. Isso tem a ver com POO também, pois não podemos sobrescrever um método desse tipo. Se voltarmos a classe Carro para não ser como final, ou seja, que possa ter classes herdeiras, e tiver um método do tipo *final*, significa que suas classes irão herdar mas não poderão modifica-las.

|  |
| --- |
| **public** **class** Carro  { **public** **static** **final** **double** ***VELOCIDADE\_LIMITE*** = 130;  **private** String nome;  **private** String marca;  **private** **final** Comprador comprador = **new** Comprador();    **public** String getNome() {**return** nome;}  **public** **void** setNome(String nome) {**this**.nome = nome;}    **public** String getMarca() {**return** marca;}  **public** **void** setMarca(String marca) {**this**.marca = marca;}    **public** Comprador getComprador() {**return** comprador;}    **public** **final** **void** descreverCarro() {**this**.toString();}    @Override **public** String toString()  {**return** "Carro [nome=" + nome + ", marca=" + marca + "]";}  } |



**Figura – Erro de compilação. Tentativa de sobreescrita de método do tipo *final*.**

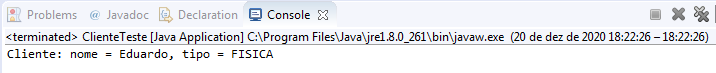
# ENUM

Enumeração foi um recurso criado a partir da versão 5 do JAVA e iremos explicar o por que. Imagine que temos a seguinte classe de cliente.

|  |
| --- |
| **public** **class** Cliente  { **private** String nome;  **public** **static** **final** **int** ***PESSOA\_FISICA*** = 1;  **public** **static** **final** **int** ***PESSOA\_JURIDICA*** = 2;  **private** **int** tipo;    **public** Cliente(String nome, **int** tipo)  { **this**.nome = nome;  **this**.tipo = tipo;  }    **public** String getNome() {**return** nome;}  **public** **void** setNome(String nome) {**this**.nome = nome;}    **public** **int** getTipo() {**return** tipo;}  **public** **void** setTipo(**int** tipo) {**this**.tipo = tipo;}  @Override **public** String toString()  { String tipoPessoa = "";  **if**(**this**.getTipo() == 1) {tipoPessoa = "FISICA";}  **else** {tipoPessoa = "JURIDICA";}  **return** "Cliente: nome = " + **this**.nome +  ", tipo = " + tipoPessoa;  }  } |

Agora iremos, além de criar uma classe de tipo ClienteTeste.java, instanciar um tipo de cliente pessoa física.

|  |
| --- |
| **public** **class** ClienteTeste  { **public** **static** **void** main(String[] args)  { //definindo um cliente tipo pessoa fisica  Cliente cliente = **new** Cliente("Eduardo",  Cliente.***PESSOA\_FISICA***);    System.***out***.println(cliente.toString());  }  } |



**Figura – Printando um Cliente.**

Pode acontecer de uma pessoa não passar mais utilizar essas variáveis estáticas constantes. A variável inteira “tipo” da classe Cliente é do tipo int, então, o que impediria um programador de digitar e passar um valor 10 por exemplo. No nosso caso, o método *toString()* da classe Cliente irá imprimir o cliente como Pessoa Jurídica, mas se tivermos que salvar essa informação no banco de dados? Esse valor inteiro poderia trazer muitos problemas. Para evitar esse tipo de situação que foi criado o *enum*, ou seja, enumerações. Nas enumerações, tudo é considerado constante.

Temos duas formas de criar uma enumeração:

## A) Criando classes:

Criar classe de forma separada, que muitos consideram a melhor opção.

|  |
| --- |
| **public** **enum** EnumCliente  {***PESSOA\_FISICA***, ***PESSOA\_JURIDICA***;} |
| **public** **class** Cliente  { **private** String nome;  //public static final int PESSOA\_FISICA = 1;  //public static final int PESSOA\_JURIDICA = 2;  **private** EnumCliente tipo;    **public** Cliente(String nome, EnumCliente tipo)  { **this**.nome = nome;  **this**.tipo = tipo;  }    **public** String getNome() {**return** nome;}  **public** **void** setNome(String nome) {**this**.nome = nome;}    **public** EnumCliente getTipo() {**return** tipo;}  **public** **void** setTipo(EnumCliente tipo) {**this**.tipo = tipo;}  @Override **public** String toString()  { String tipoPessoa = "";  **if**(**this**.getTipo() == EnumCliente.***PESSOA\_FISICA***)  {tipoPessoa = "FISICA";}  **else** {tipoPessoa = "JURIDICA";}  **return** "Cliente: nome = " + **this**.nome +  ", tipo = " + tipoPessoa;  }  } |
| **import** br.edu.maratona.enumeracao.EnumCliente;  **public** **class** ClienteTeste  { **public** **static** **void** main(String[] args)  { //definindo um cliente tipo pessoa fisica  Cliente cliente = **new** Cliente("Eduardo",  EnumCliente.***PESSOA\_FISICA***);    System.***out***.println(cliente.toString());  }  } |

Dessa forma, ao invés de um programador ou qualquer profissional da área que tentasse implementar algo, o impediria de escrever qualquer coisa, pois ao invés de permiti-lo de escrever um valor inteiro, obrigará escrever um tipo constante.

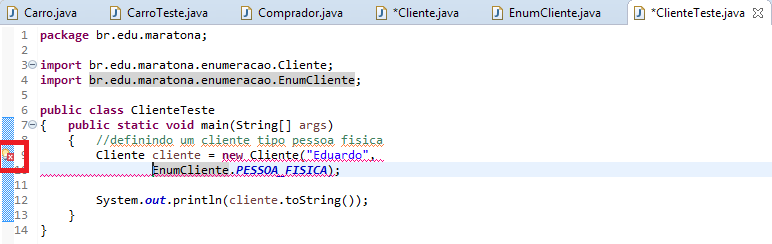
## B) Criar como Atributo de classe:

Como um *enum* é um tipo constante, basta então criarmos uma variável local de classe do tipo *enum*, definindo valores em sequência. Então podemos simplesmente fazer uma cópia do que escrevemos na classe *EnumCliente* e colar na classe Cliente.

|  |
| --- |
| **public** **class** Cliente  { **private** String nome;  //public static final int PESSOA\_FISICA = 1;  //public static final int PESSOA\_JURIDICA = 2;  **public** **enum** EnumCliente {***PESSOA\_FISICA***, ***PESSOA\_JURIDICA***;}  **private** EnumCliente tipo;      **public** Cliente(String nome, EnumCliente tipo)  { **this**.nome = nome;  **this**.tipo = tipo;  }    **public** String getNome() {**return** nome;}  **public** **void** setNome(String nome) {**this**.nome = nome;}    **public** EnumCliente getTipo() {**return** tipo;}  **public** **void** setTipo(EnumCliente tipo) {**this**.tipo = tipo;}  @Override **public** String toString()  { String tipoPessoa = "";  **if**(**this**.getTipo() == EnumCliente.***PESSOA\_FISICA***)  {tipoPessoa = "FISICA";}  **else** {tipoPessoa = "JURIDICA";}  **return** "Cliente: nome = " + **this**.nome +  ", tipo = " + tipoPessoa;  }  } |

O problema de fazer dessa maneira, é que toda vez que tivermos que chamar na classe de execução, além de chamar o tipo EnumCliente, primeiro temos que chamar a classe Cliente para depois chamar o tipo *enum*. Caso contrário, teremos um erro de compilação.

|  |
| --- |
| **public** **class** ClienteTeste  { **public** **static** **void** main(String[] args)  { //definindo um cliente tipo pessoa fisica  Cliente cliente = **new** Cliente("Eduardo",  Cliente.EnumCliente.***PESSOA\_FISICA***);    System.***out***.println(cliente.toString());  }  } |



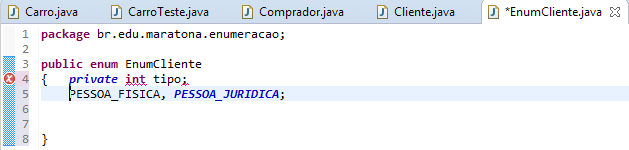
**Figura – Erro de compilação: Atributo tipo *enum* numa classe.**

**OBS: De maneira alguma é permitido criar um *enum* em um método no java.**

Enumeração é um tipo especial de classe, pois podemos até mesmo adicionar métodos dentro de um *enum*. Então voltando para o nosso EnumCliente, lembra que antes de tê-la criada, tínhamos somente um inteiro “tipo” que só admitiam valores 1 e 2? Imagine que estamos fazendo modificação em um sistema, onde banco de dados já estava salvando esses valores e precisamos continuar salvando. Para resolver esse problema, podemos fazer com que a nossa enumeração tenha um construtor e tenha atributos de classe e métodos também. Nunca poderemos chamar ou inicializar um construtor de uma enumeração, pois ele próprio será chamado.

Então podemos criar atributos, porém existe uma regra: não podemos criar antes da própria enumeração.

|  |
| --- |
| **public** **enum** EnumCliente  { ***PESSOA\_FISICA***, ***PESSOA\_JURIDICA***;    **private** **int** tipo;  } |

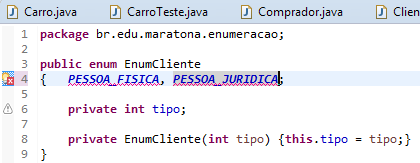


**Figura – Erro de compilação: Atributos em enum.**

Para criar um construtor, jamais podemos defini-lo como *public*, pois nunca devemos instanciar um tipo *enum*, pois ele mesmo será chamado. Além disso, devemos definir um numero para os parâmetros:

|  |
| --- |
| **public** **enum** EnumCliente  { ***PESSOA\_FISICA***(1), ***PESSOA\_JURIDICA***(2);    **private** **int** tipo;  **private** EnumCliente(**int** tipo) {**this**.tipo = tipo;}  } |

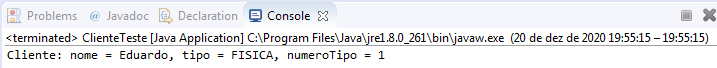
Caso contrário teremos um erro de compilação, como vemos na figura abaixo.



**Figura – Erro de compilação: Construtor em *enum*.**

Agora iremos inserir um método, por exemplo, tipo GETTER somente, pois se um tipo SETTER fosse possível, ou seja métodos modificadores de acesso, deixaria de ser um *enum*.

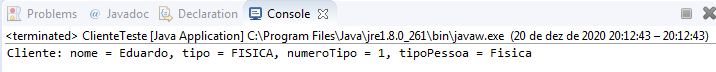
|  |
| --- |
| **public** **enum** EnumCliente  { ***PESSOA\_FISICA***(1), ***PESSOA\_JURIDICA***(2);    **private** **int** tipo;  **private** EnumCliente(**int** tipo) {**this**.tipo = tipo;}  **public** **int** getTipo() {**return** tipo;}  } |
| **public** **class** Cliente  { **private** String nome;  //public static final int PESSOA\_FISICA = 1;  //public static final int PESSOA\_JURIDICA = 2;  //public enum EnumCliente {PESSOA\_FISICA, PESSOA\_JURIDICA;}  **private** EnumCliente tipo;    **public** Cliente(String nome, EnumCliente tipo)  { **this**.nome = nome;  **this**.tipo = tipo;  }    **public** String getNome() {**return** nome;}  **public** **void** setNome(String nome) {**this**.nome = nome;}    **public** EnumCliente getTipo() {**return** tipo;}  **public** **void** setTipo(EnumCliente tipo) {**this**.tipo = tipo;}  @Override **public** String toString()  { String tipoPessoa = "";  **if**(**this**.getTipo() == EnumCliente.***PESSOA\_FISICA***)  {tipoPessoa = "FISICA";}  **else** {tipoPessoa = "JURIDICA";}  **return** "Cliente: nome = " + **this**.nome +  ", tipo = " + tipoPessoa + ", numeroTipo = " +  **this**.tipo.getTipo();  }  } |



**Figura – Resultado do método tipo GETTER em um enum.**

Podemos também, no caso dos construtores, inserir parâmetros de outro tipo, por exemplo string, para indicar que valor real possui o atributo “tipo” definido.

|  |
| --- |
| **public** **enum** EnumCliente  { ***PESSOA\_FISICA***(1, "Fisica"), ***PESSOA\_JURIDICA***(2, "Juridica");    **private** **int** tipo;  **private** String tipoPessoa;    **private** EnumCliente(**int** tipo, String tipoPessoa)  { **this**.tipo = tipo;  **this**.tipoPessoa = tipoPessoa;  }  **public** **int** getTipo() {**return** tipo;}  **public** String getTipoPessoa() {**return** tipoPessoa;}  } |
| **public** **class** Cliente  { **private** String nome;  //public static final int PESSOA\_FISICA = 1;  //public static final int PESSOA\_JURIDICA = 2;  //public enum EnumCliente {PESSOA\_FISICA, PESSOA\_JURIDICA;}  **private** EnumCliente tipo;    **public** Cliente(String nome, EnumCliente tipo)  { **this**.nome = nome;  **this**.tipo = tipo;  }    **public** String getNome() {**return** nome;}  **public** **void** setNome(String nome) {**this**.nome = nome;}    **public** EnumCliente getTipo() {**return** tipo;}  **public** **void** setTipo(EnumCliente tipo) {**this**.tipo = tipo;}  @Override **public** String toString()  { String tipoPessoa = "";  **if**(**this**.getTipo() == EnumCliente.***PESSOA\_FISICA***)  {tipoPessoa = "FISICA";}  **else** {tipoPessoa = "JURIDICA";}  **return** "Cliente: nome = " + **this**.nome +  ", tipo = " + tipoPessoa + ", numeroTipo = " +  **this**.tipo.getTipo() + ", tipoPessoa = " +  **this**.tipo.getTipoPessoa();  }  } |



**Figura – Resultado de um *enum* com outro tipo de atributo.**

Resumindo, as enumerações são muito úteis quando queremos definir constantes, em que os valores que não serão e nem deverão serem mudados.