**Capítulo CINCO**

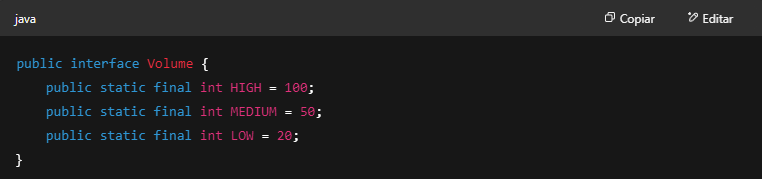
**Enumerações**

**Objetivos do Exame**

Utilizar tipos enumerados incluindo métodos e construtores em um tipo enum.

**Enumerações**

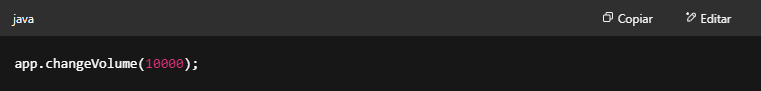
Digamos que nossa aplicação use três estados ou valores para o volume de reprodução de uma música:  
**alto**, **médio**, **baixo**.  
Normalmente modelamos isso com algumas “constantes”, como esta:



No entanto, esta não é uma boa implementação. Considere um método para alterar o volume:



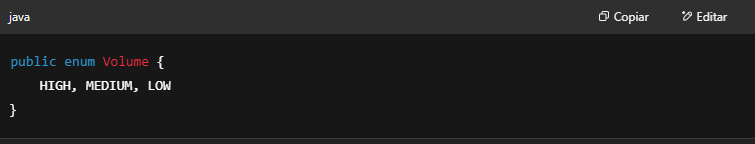
O que impede alguém de chamar esse método com um valor arbitrário como:



Claro, podemos implementar algumas verificações, mas isso apenas torna o problema mais complicado.  
Felizmente, enumerações (ou **enums**, para abreviar) oferecem uma boa solução para esse problema.

Um enum é um tipo (como uma classe ou interface) que representa uma **lista FIXA de valores**  
(podemos pensar neles como constantes).

Portanto, podemos usar um enum para representar o volume de um som em nossa aplicação (observe o uso de enum em vez de interface):



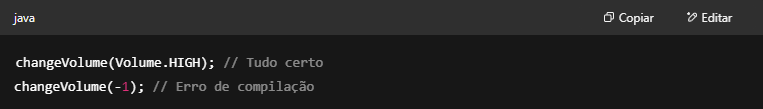
Observe que, como os valores são "constantes" (são implicitamente public, static e final), a convenção de usar **letras maiúsculas** é seguida.

Também observe que os valores são separados por vírgulas e, como o enum contém apenas uma lista de valores, o ponto e vírgula **é opcional** após o último.

Dessa forma, podemos definir o método changeVolume assim:

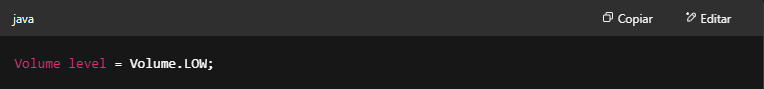


Passar qualquer outro objeto que **não seja um Volume** gerará um erro de compilação:

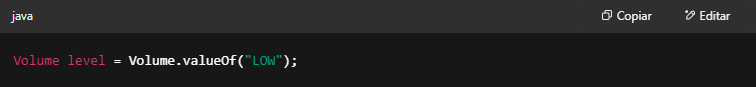


O método agora é **type-safe** (seguro quanto ao tipo), o que significa que **não podemos atribuir valores inválidos**.

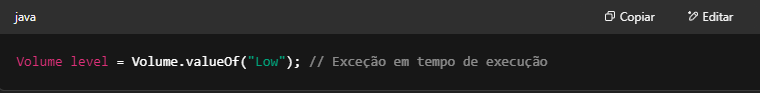
Aliás, devido a algo que veremos depois, **você não pode usar o operador new** para obter uma referência de um enum, então obtemos a referência diretamente:



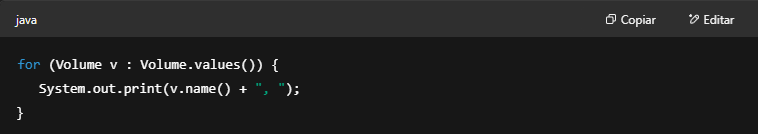
Você também pode obter um enum a partir de uma String, por exemplo:



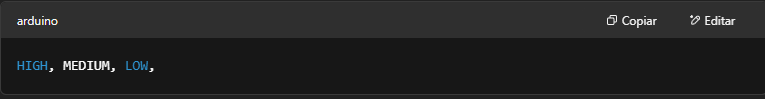
Atenção: esse método **faz distinção entre maiúsculas e minúsculas**:



Para obter todos os valores de um tipo enum use o método values(), que retorna um **array de enums** na mesma ordem em que foram declarados.  
Ele funciona muito bem com for-each:

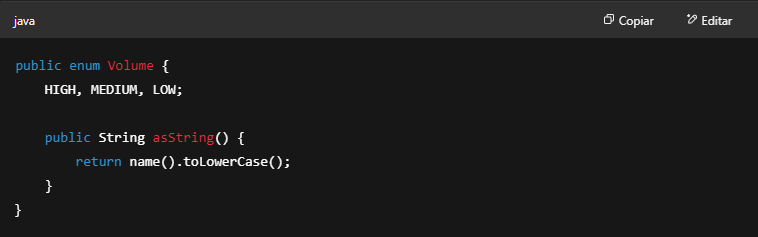


Saída:



**Trabalhando com Enums**

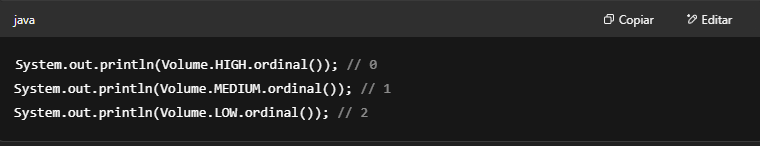
Você pode usar métodos de instância em enums, por exemplo:



O método name() retorna o nome original conforme definido na enumeração.  
O método asString() acima apenas o converte para minúsculas.

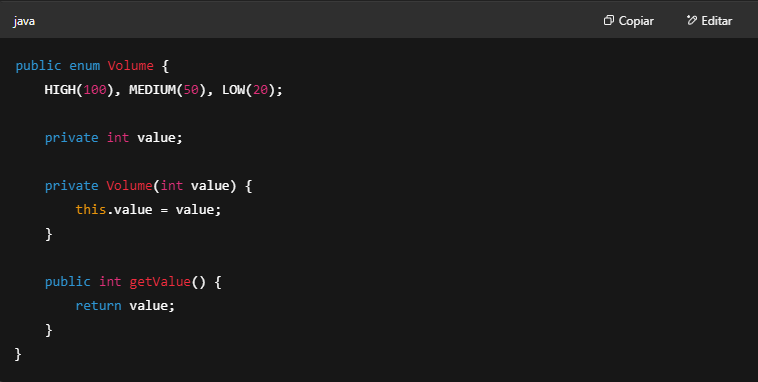
Note que o nome asString() é apenas um nome de método arbitrário.  
O método toString() também pode ser sobrescrito.

O método ordinal() retorna a **posição (índice)** do enum na lista:



Enumerações são ordenadas a partir de zero, na **ordem em que os valores foram declarados**.

Enums também podem ter **valores associados**. Isso é feito através de **construtores**, e esses construtores podem ter argumentos:



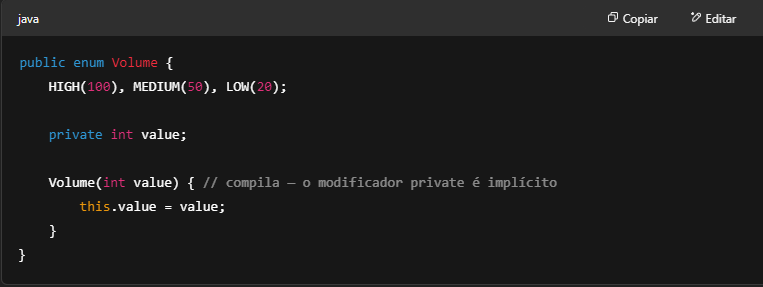
Atenção para os seguintes pontos:

* Os valores associados são definidos **na declaração do enum** (HIGH(100), etc).
* A variável value foi declarada como private int.
* O construtor também é private — e deve ser assim!
* Há um método getValue() para acessar o valor inteiro de cada enum.

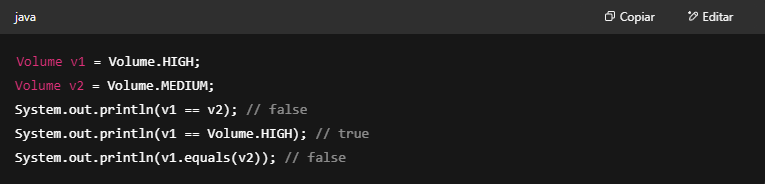
Agora podemos fazer:



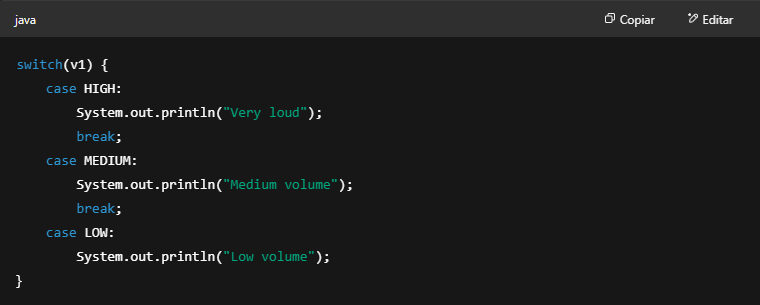
Como dito anteriormente, **não podemos criar uma instância de enum com new**, e por isso o construtor **deve ser privado**:



Você pode comparar enums com == ou equals() (o primeiro é mais comum e mais rápido, pois são referências constantes):



Você também pode usar switch com enums (a partir do Java 5):

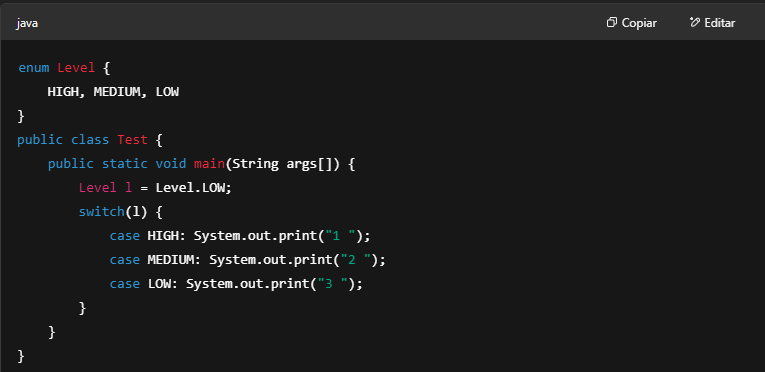


**Pontos-chave**

* Enums são tipos especiais que representam um **conjunto fixo de constantes**.
* Um tipo enum é definido com a palavra-chave enum e pode ser public ou package-private.
* Os elementos definidos em uma enum são **implícita e automaticamente**:
  + public
  + static
  + final
* Enums fornecem um tipo seguro, ou seja, **apenas valores válidos** podem ser atribuídos a variáveis desse tipo.
* A partir do Java 5, switch pode ser usado com enums.
* Enums podem conter:
  + Métodos
  + Construtores
  + Campos
  + Variáveis de instância
* Os **construtores de enum são sempre private** (implícita ou explicitamente) e **nunca são chamados diretamente**.
* O método ordinal() retorna a **posição baseada em zero** do enum na lista de declaração.
* O método values() retorna **um array de todos os enums definidos**, na ordem de declaração.
* Você pode obter um enum a partir de uma String com valueOf(String name), que lança uma exceção se o nome não for válido.
* Você pode comparar enums com ==, equals(), ou usá-los em estruturas de controle como switch.

**Autoavaliação (Self Test)**

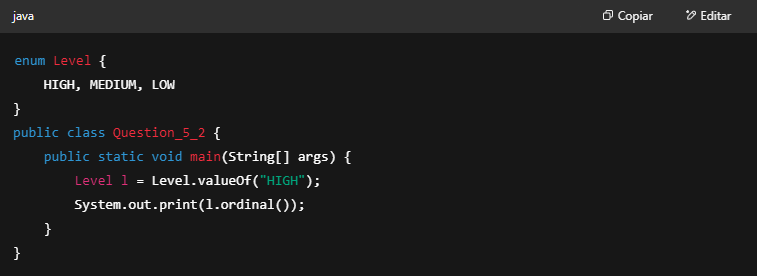
**1. Dado:**



**Qual é a saída?**

A. 1 2 3  
B. 3  
C. 2 3  
D. Nada  
E. Falha de compilação

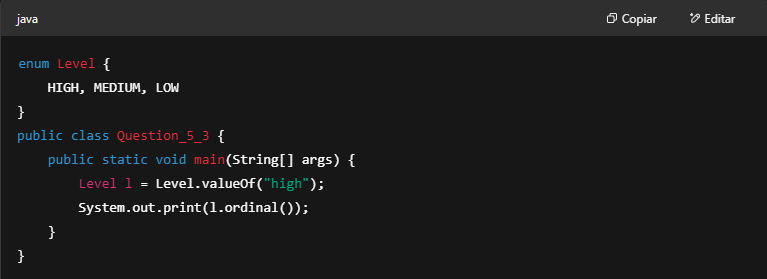
**2. Dado:**



**Qual é a saída?**

A. 0  
B. 1  
C. 2  
D. HIGH  
E. Falha de compilação  
F. Exceção em tempo de execução

**3. Dado:**



**Qual é a saída?**

A. 0  
B. 1  
C. 2  
D. HIGH  
E. Falha de compilação  
F. Exceção em tempo de execução

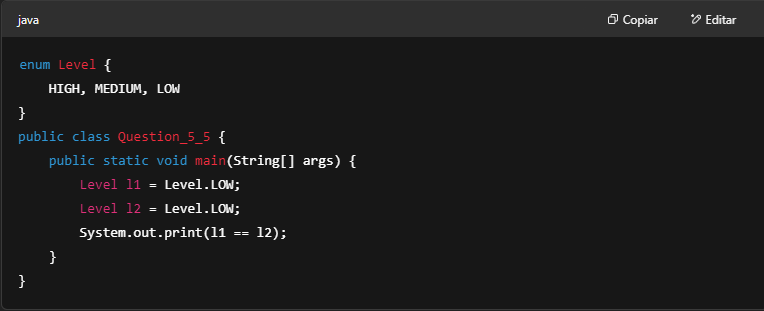
**4. Dado:**



**Qual é a saída?**

A. 3  
B. 2  
C. 1  
D. Falha de compilação  
E. Exceção em tempo de execução

**5. Dado:**



**Qual é a saída?**

A. true  
B. false  
C. Falha de compilação  
D. Exceção em tempo de execução