**Capítulo TRÊS**

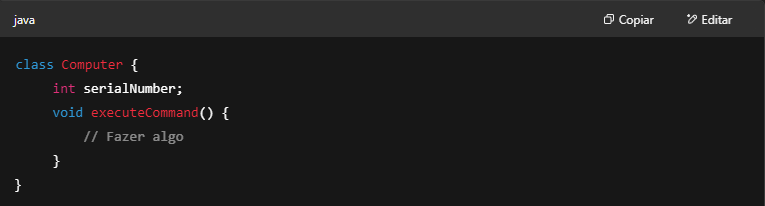
**Inner Classes**  
**Objetivos do Exame**  
Criar classes internas, incluindo classes internas estáticas, classes locais, classes aninhadas e classes internas anônimas.

**Classes**

Em Java temos classes:



As classes têm dois tipos de membros, atributos (ou campos) e métodos (ou funções):



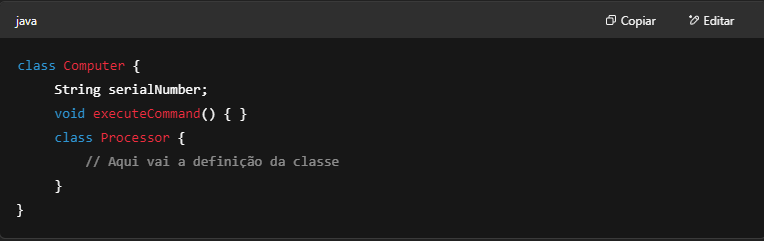
Dessa forma, nossos programas são uma coleção de classes:



**Classes Internas**

O Java nos dá flexibilidade na forma como podemos projetar nossas classes.

Para isso, há um terceiro tipo de membro que uma classe pode ter: uma **CLASSE INTERNA**.

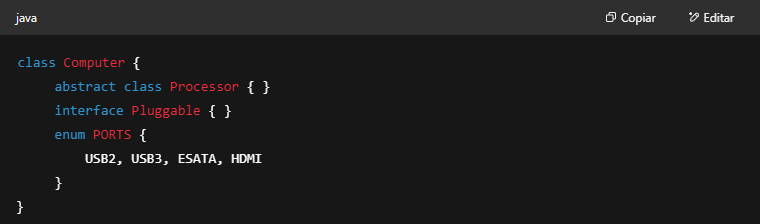


Classes internas também são conhecidas como **classes aninhadas**. Em teoria, você pode ter muitos níveis de classes.

Tenho dificuldade em pensar em qual seria o benefício de ter mais de um nível de classes internas:



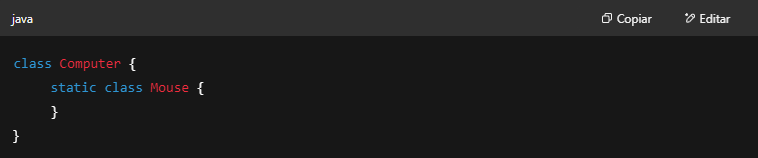
Outra coisa. Sempre falamos sobre CLASSES internas, mas na verdade, podemos ter CLASSES ABSTRATAS internas, INTERFACES internas e ENUMS internos:



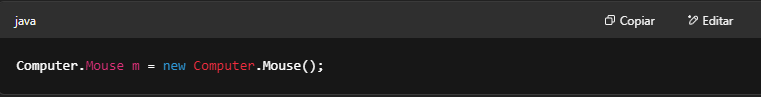
Mas vamos focar em classes internas simples. Existem quatro tipos delas:

* Classes internas **estáticas**
* Classes internas **não estáticas**
* **Classes locais**
* **Classes anônimas**

**Definindo uma classe interna estática**



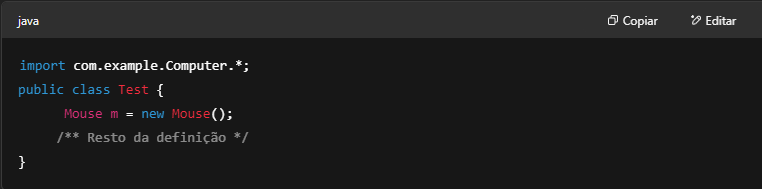
Usando uma classe interna estática



Classes internas estáticas são acessadas por meio de sua classe envolvente.

Classes estáticas são **INDEPENDENTES** da sua classe envolvente. Elas são como classes comuns, apenas por estarem dentro de outra classe.

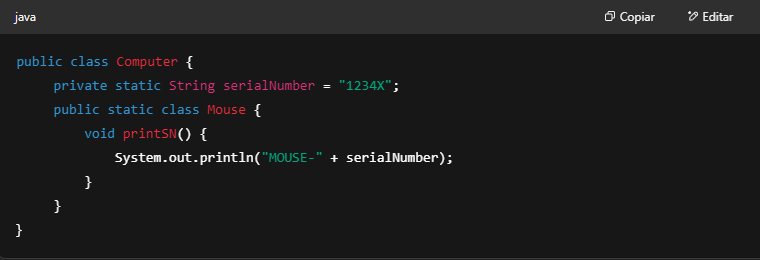
Na verdade, você pode pensar na classe envolvente como um tipo de pacote. Você pode importar o nome da classe envolvente e usar a classe interna estática como uma classe normal. Apenas lembre-se de que a classe interna estática deve ser um membro **público** para ser acessada de outro pacote.



E elas também podem ser marcadas como private, protected ou sem modificador, então são acessíveis apenas no pacote (acessibilidade padrão).



Claro que, sendo um membro de uma classe, a classe interna estática tem acesso a outros membros da classe envolvente, mas **somente se eles forem STATIC**.



**Por quê?**

Pense nisso: se a classe estática é independente da classe envolvente, ela não precisa de uma instância desta, então apenas os membros static podem ser usados, pois estão associados à classe, não a uma instância específica.

Por essa razão, uma classe interna estática é frequentemente usada como uma **classe utilitária**, que contém métodos comuns compartilhados por todos os objetos de uma classe.

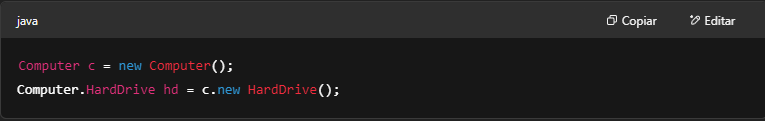
Se você usar a classe interna estática dentro da classe que a define, você pode usá-la em qualquer método, bloco ou construtor, **não importa se for estático ou não**, já que a classe interna não está vinculada a uma instância específica.

**Definindo uma classe interna não estática**



**Usando uma classe interna não estática**

**Classes internas não estáticas** são acessadas por meio de uma instância da sua classe envolvente:

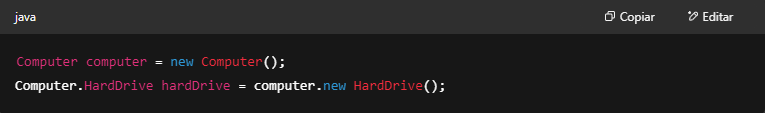


Observe o tipo da classe interna e como o operador new é usado.

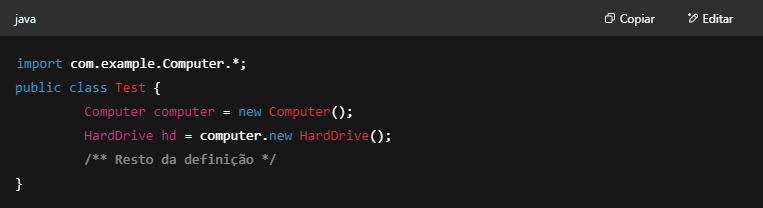
Classes internas não estáticas são simplesmente chamadas de **classes internas**.

Instâncias de uma classe interna existem apenas **DENTRO** de uma instância da classe envolvente. É o mesmo que quando você quer usar um método de uma classe, você PRIMEIRO precisa de uma instância dessa classe.

Uma vez que você tenha uma instância da classe envolvente, você usa o operador new de uma maneira (estranha) diferente da que normalmente se usa:



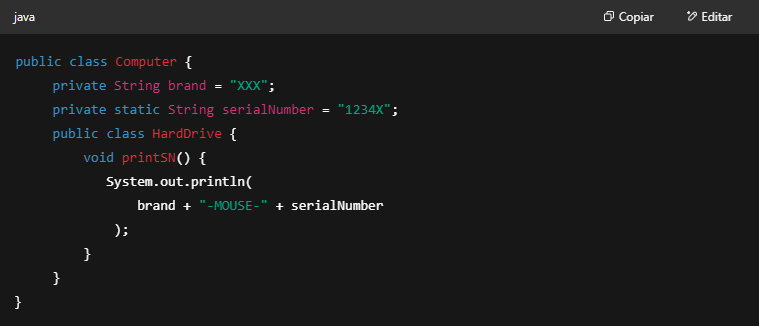
Você também pode usar o truque de importação para escrever menos, mas ainda assim precisa criar a classe interna como sempre:



Outra forma de obter uma instância de uma classe interna é usar um método da classe envolvente para criá-la, evitando essa sintaxe estranha:



Por ser um membro de uma classe, a classe interna tem acesso aos outros membros da classe envolvente, mas desta vez, **NÃO IMPORTA** se são static ou não:



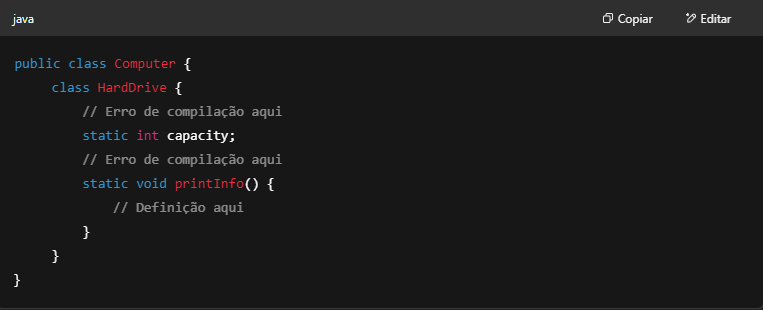
**Por quê?**

Porque para usar a classe interna, **uma instância da classe envolvente é exigida**, garantindo que os membros não estáticos existam (membros static podem ser acessados de qualquer forma).

Classes internas também podem ser marcadas como private, protected ou sem modificador, então são acessíveis apenas no pacote. Mas na maioria das vezes, como elas dependem da classe envolvente, são marcadas como private.



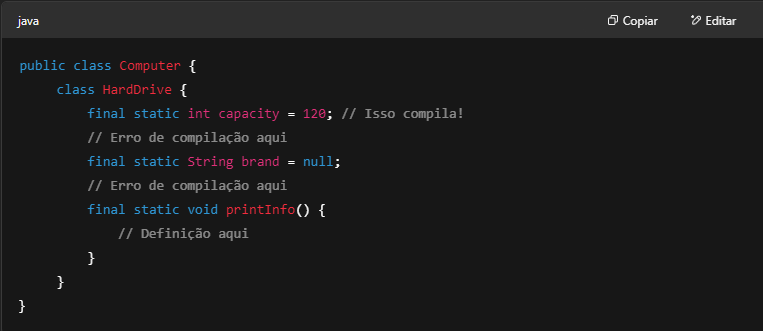
Outra regra é que **classes internas NÃO PODEM conter membros static**:



Código static é executado durante a inicialização da classe, mas você não pode inicializar uma classe interna não estática sem ter uma instância da classe envolvente.

Porque uma classe interna pertence a UMA instância da classe envolvente. Ter um membro static significa que ele pode ser compartilhado entre instâncias, pois o membro **pertence à classe**, mas como estamos falando de uma classe interna que **não pode ser compartilhada** entre várias instâncias da classe envolvente, isso não é possível.

A única exceção é quando você define um atributo final static. A palavra-chave final faz toda a diferença; ela define uma expressão constante, mas só funciona com **atributos** e ao atribuir um valor **NÃO NULO**:



**Definindo uma classe local**

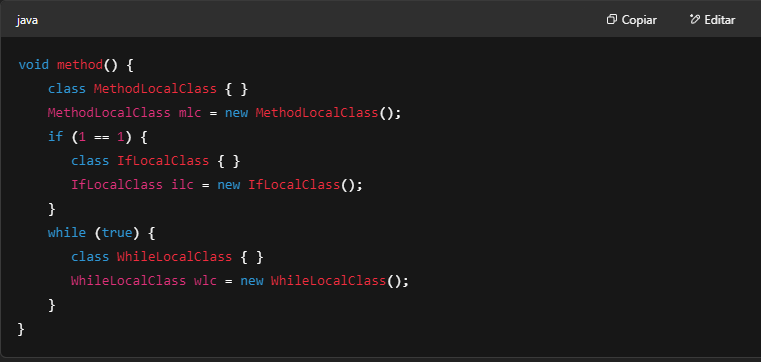


**Usando uma classe local**

**Classes locais só podem ser usadas dentro do método ou bloco que as define**:



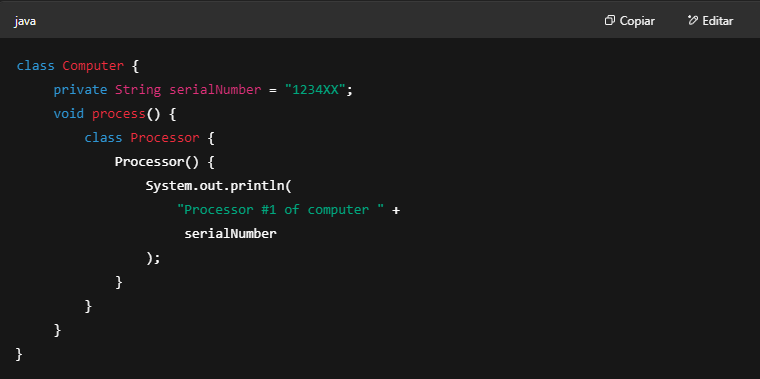
Classes locais são chamadas assim porque **só podem ser usadas no método ou bloco onde são declaradas**. Blocos são, praticamente, qualquer coisa entre chaves {}:



Observe também **onde as instâncias das classes locais são criadas**. A classe local deve ser usada **ABAIXO** de sua definição. Caso contrário, o compilador não conseguirá encontrá-la.

Como uma classe interna local **não é membro de uma classe**, ela **NÃO pode ser declarada com nível de acesso**, e isso não faria sentido mesmo, já que são acessíveis apenas no local onde são declaradas. No entanto, uma classe local pode ser declarada como abstract ou final (mas **não ao mesmo tempo**).

Classes locais **exigem uma instância da sua classe envolvente** para que o método ou bloco onde estão definidas possa ser executado. Por essa razão, elas podem acessar os membros da classe envolvente, mas **não podem declarar membros static** (apenas **atributos static final**), assim como as classes internas.



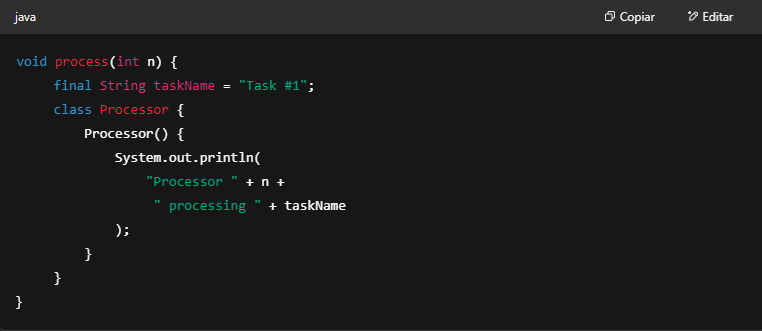
Se a classe local for declarada dentro de um método, ela pode acessar as variáveis e parâmetros do método **APENAS se forem declarados como final ou forem efetivamente finais**.

“Efetivamente final” é um termo que significa que uma variável ou parâmetro **não é alterado após ser inicializado**, mesmo que sua declaração **não use a palavra-chave final**.

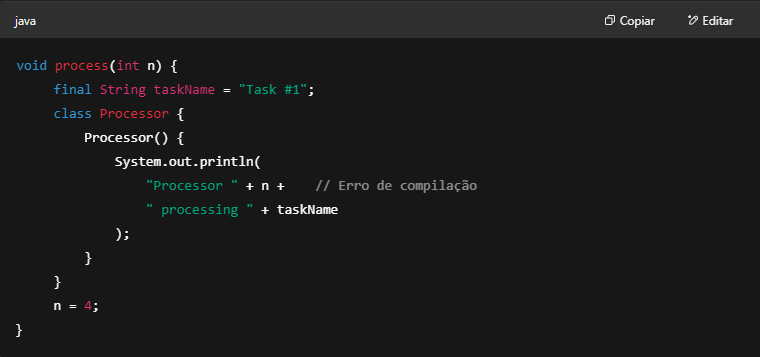
**Por quê?**

Porque uma instância de uma classe local pode continuar existindo **mesmo depois que o método ou bloco onde ela foi definida terminar sua execução** (por exemplo, se uma referência for salva em um objeto com escopo maior). Por isso, a classe local precisa manter uma **cópia interna** das variáveis que usa, e a **única forma de garantir que ambas as cópias tenham sempre o mesmo valor** é tornando a variável final.

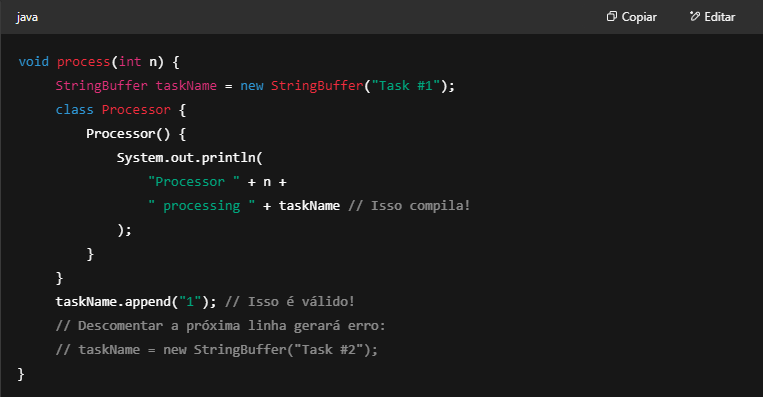
Portanto, o seguinte código é válido porque taskName é declarado como final enquanto n **não muda** e é considerado efetivamente final:



Mas se modificarmos o valor de n em algum lugar, um erro será gerado:



“Efetivamente final” só se preocupa com **referências**, não com objetos ou seu conteúdo, porque no fim das contas, estamos nos referindo ao mesmo objeto:

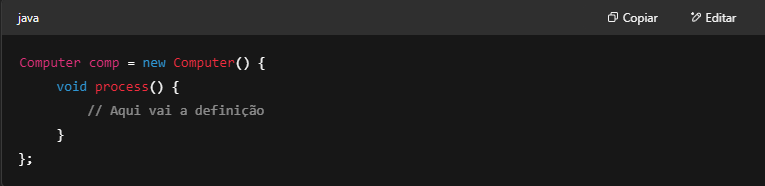


Se você ainda não tiver certeza se uma declaração é efetivamente final, tente adicionar o modificador final a ela. Se o programa continuar funcionando da mesma maneira, então a declaração é efetivamente final.

Se a classe for declarada dentro de um **método estático**, as regras de static também se aplicam, ou seja, a classe local só terá acesso aos membros static da classe envolvente.

**Definindo uma classe anônima**

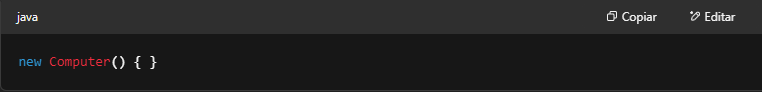
O operador new é seguido pelo nome de uma interface ou de uma classe e pelos argumentos para um construtor (ou parênteses vazios se for uma interface):



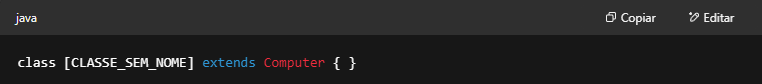
Veja como termina com um ponto e vírgula, como qualquer outro comando (statement) em Java.

O corpo da classe **implementa a interface** ou **estende a classe** referenciada.

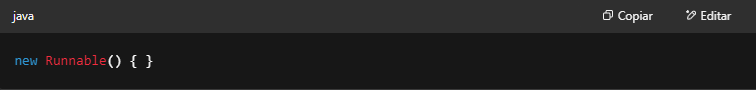
Uma **classe anônima** é chamada assim porque **não tem nome**. No entanto, uma expressão de classe anônima **não declara uma nova classe**. Ela **implementa uma interface existente** ou **estende uma classe existente**. Então:



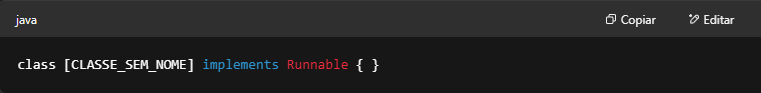
é como escrever:



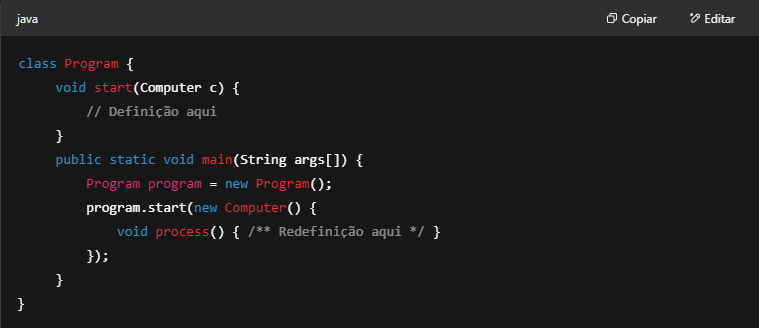
E se estivermos trabalhando com uma interface:



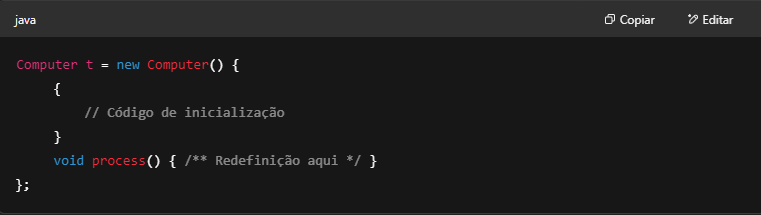
é como escrever:



Além disso, uma classe anônima pode ser usada **em uma declaração ou em uma chamada de método**:



Como elas não têm nome (bem, na verdade o compilador dá um nome aleatório a elas quando cria o arquivo .class), classes anônimas **não podem ter CONSTRUTORES**. Se você quiser executar algum código de inicialização, **tem que fazer isso com um bloco de inicialização**:



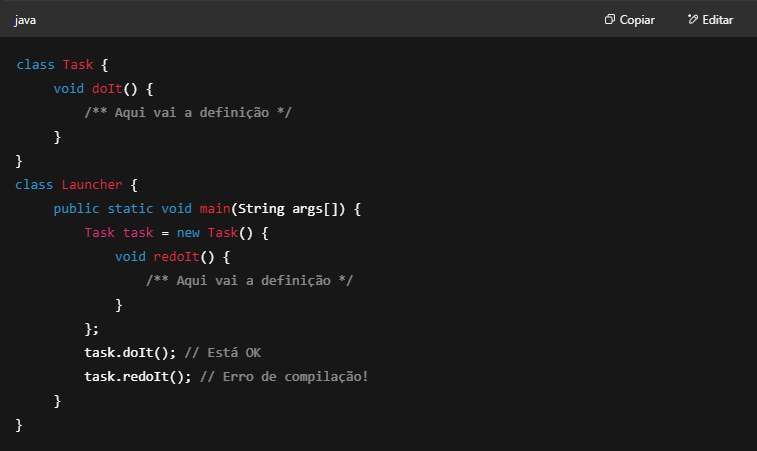
Como as classes anônimas são um tipo de **classe local**, elas seguem as mesmas regras:

* ✅ Podem acessar os membros da sua classe envolvente
* ❌ Não podem declarar membros static (exceto variáveis static final)
* ✅ Só podem acessar variáveis locais (variáveis ou parâmetros definidos em um método) **se forem final ou efetivamente finais**

**Mas tem algo com o qual você precisa ter cuidado: herança.**

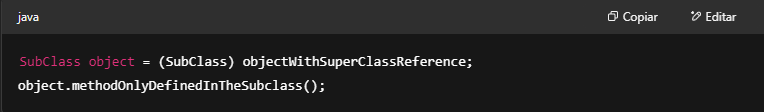
Quando você usa uma classe anônima (um objeto de uma subclasse), você está usando uma referência da superclasse. Com essa referência, você pode usar os **atributos e métodos declarados nesse tipo**.

Mas o que acontece quando você **declara um novo método na classe anônima**?



O programa irá falhar. A referência **não conhece** o método redoIt() porque ele **não está definido na superclasse**.

Tipicamente, você faria um cast para o tipo da subclasse onde o novo método está definido:



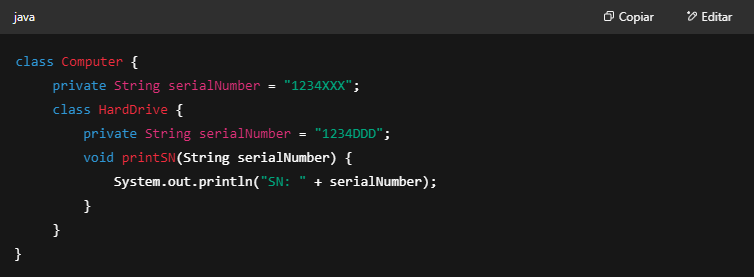
Mas com uma classe anônima, como o cast seria feito?

**Não pode ser feito**; a classe **não tem nome**, então **não há como** usarmos os métodos definidos na declaração da classe anônima. Só podemos usar os métodos definidos na **SUPERCLASSE** (seja ela uma interface ou classe).

Usar uma classe anônima é, na maioria das vezes, uma questão de estilo. Se a classe tem um corpo curto, **implementa apenas uma interface** (se estamos trabalhando com interfaces), **não declara novos membros**, e a sintaxe deixa seu código mais claro, você deve considerar usá-la em vez de uma classe local ou interna.

**Sombreamento (Shadowing)**

Um conceito importante a ser levado em consideração ao se trabalhar com classes internas (de qualquer tipo) é o que acontece quando **um membro da classe interna tem o mesmo nome de um membro da classe envolvente**:



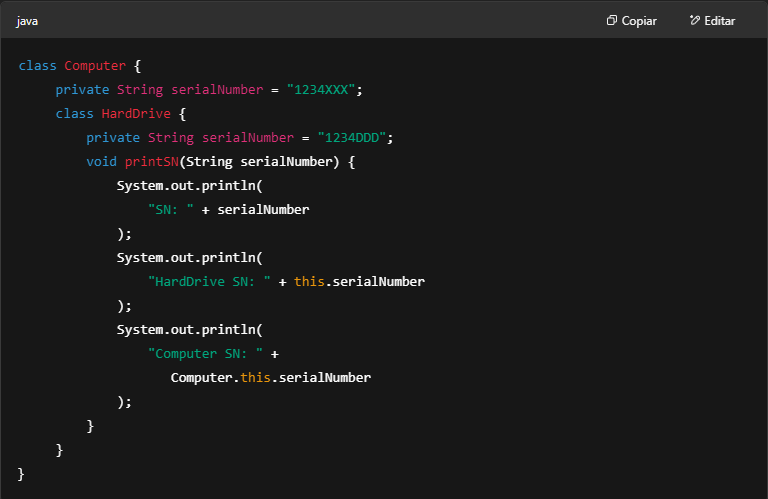
Nesse caso, o parâmetro serialNumber **sombreia** a variável de instância serialNumber da classe HardDrive, que por sua vez **sombreia** o serialNumber da classe Computer.

Do jeito que está escrito, o método printSN() imprimirá o argumento. **Uma declaração sombreada precisa de algo a mais para ser referida corretamente.**

Sabemos que, quando um objeto quer se referir a si mesmo, usamos a palavra-chave this.

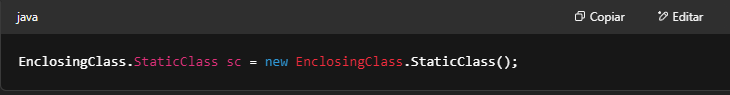
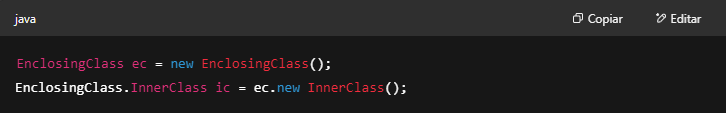
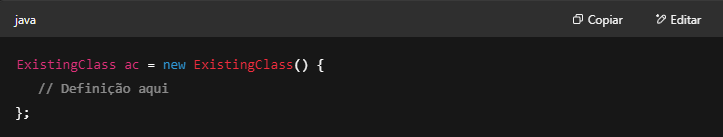
Então, se usarmos this **dentro de uma classe interna**, ele irá se referir **à própria classe interna**.

Se precisarmos referenciar a **classe envolvente**, **dentro da classe interna também podemos usar this**, mas da seguinte forma: NomeDaClasseEnvolvente.this.



Como isso pode causar confusão, é melhor evitar e **usar nomes de variáveis descritivos**.

**Pontos-chave (Key Points)**

* Classes internas são declaradas dentro de outra classe. Há quatro tipos delas:
  + **Classes estáticas**
  + **Classes não estáticas**
  + **Classes locais**
  + **Classes anônimas**
* **Classes estáticas** são apenas classes internas marcadas com a palavra-chave static. No entanto, elas se comportam mais como uma **classe de nível superior** do que como uma classe interna.
* Você **não precisa de uma instância da classe envolvente para instanciar uma classe estática**:
* Uma classe estática **não pode acessar membros não estáticos** da classe envolvente, pois **não precisa de uma instância** da classe envolvente para ser usada.
* Uma **classe interna (não estática)** é como qualquer outro membro da classe envolvente, podendo ser marcada com qualquer modificador de acesso.
* Fora dos métodos de instância da classe envolvente, para instanciar uma classe interna, você deve primeiro criar uma instância da classe envolvente, e então:
* Uma **classe local** é definida dentro de um método ou bloco da classe envolvente.
* Os únicos modificadores que se aplicam a uma classe local são abstract e final (mas **não ao mesmo tempo**).
* Você só pode usar uma classe local **no método ou bloco onde ela é definida**, e **somente após sua declaração**.
* Uma classe local pode acessar os membros da classe, como qualquer outro membro (as regras de static ainda se aplicam).
* No entanto, uma classe local **só pode acessar os parâmetros e variáveis locais** de um método se forem final ou **efetivamente finais**.
* **Efetivamente final** significa que uma variável não pode ser modificada após sua inicialização, **mesmo que não esteja marcada explicitamente como final**.
* **Classes anônimas** não têm nome e **estendem uma classe existente ou implementam uma interface**:
* Classes anônimas **não podem ter construtores**.
* Classes anônimas seguem as mesmas regras das classes locais quanto ao acesso aos membros da classe envolvente e às variáveis locais de um método.
* Os únicos métodos que você pode chamar em uma classe anônima são aqueles **definidos no tipo de referência** (a superclasse ou a interface), **mesmo que a classe anônima defina seus próprios métodos**.

**Autoavaliação (Self Test)**

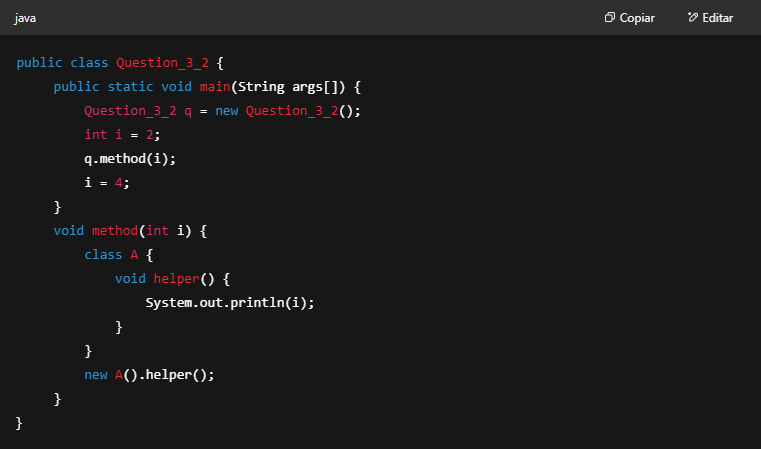
**1. Dado:**



**Qual é o resultado?**

A. m()  
B. Falha de compilação na declaração marcada como // 1  
C. Falha de compilação na declaração marcada como // 2  
D. Uma exceção ocorre em tempo de execução

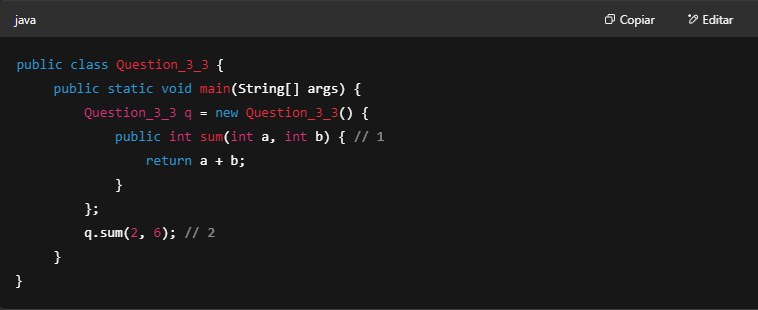
**2. Dado:**



**Qual é o resultado?**

A. Falha de compilação  
B. 2  
C. 4  
D. Uma exceção ocorre em tempo de execução

**3. Dado:**



**Qual é o resultado?**

A. Falha de compilação na declaração marcada como // 1  
B. Falha de compilação na linha marcada como // 2  
C. 8  
D. Nada é impresso

**4. Dado:**



**Qual é o resultado?**

A. Falha de compilação porque uma classe interna não pode ser static  
B. Falha de compilação porque a classe Inner foi instanciada incorretamente dentro do método main  
C. Falha de compilação porque o método doIt não pode ser chamado em main por ser declarado como private  
D. O programa imprime doIt()

**5. Dado:**



**Qual é o resultado?**

A. Falha de compilação primeiro na linha // 1  
B. Falha de compilação primeiro na linha // 2  
C. Falha de compilação na linha // 3  
D. go! é impresso

**6. Dado:**



**Qual é o resultado?**

A. Falha de compilação na linha // 1  
B. Falha de compilação na linha // 2  
C. 5  
D. Uma exceção ocorre em tempo de execução

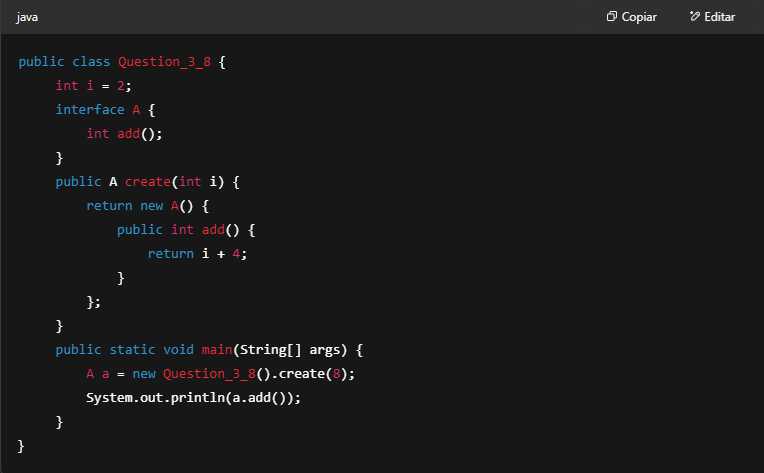
**7. Dado:**



**Qual mudança faria este código compilar?**

A. Remover a palavra-chave abstract na linha // 1  
B. Adicionar a palavra-chave public na linha // 1  
C. Remover a palavra-chave public na linha // 2  
D. Nenhuma. Este código compila corretamente

**8. Dado:**



**Qual é o resultado?**

A. 6  
B. 12  
C. Falha de compilação  
D. Uma exceção ocorre em tempo de execução