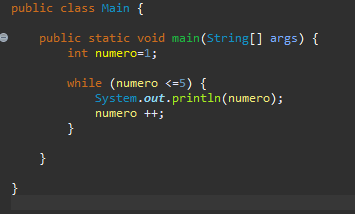
**Instrutura de repetição:**

Usado para repetir algo, ex: enquanto for menor que 10 faça isso...

* **While:**

Estrutura **While (condição) {...}**



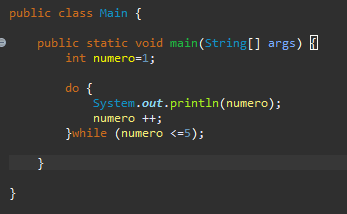
Não é usado para contar vezes! Geralmente utiliza-se o for!

**Normalmente usado quando não sei a quantidade de vezes vai usar**!

* **Do while:**

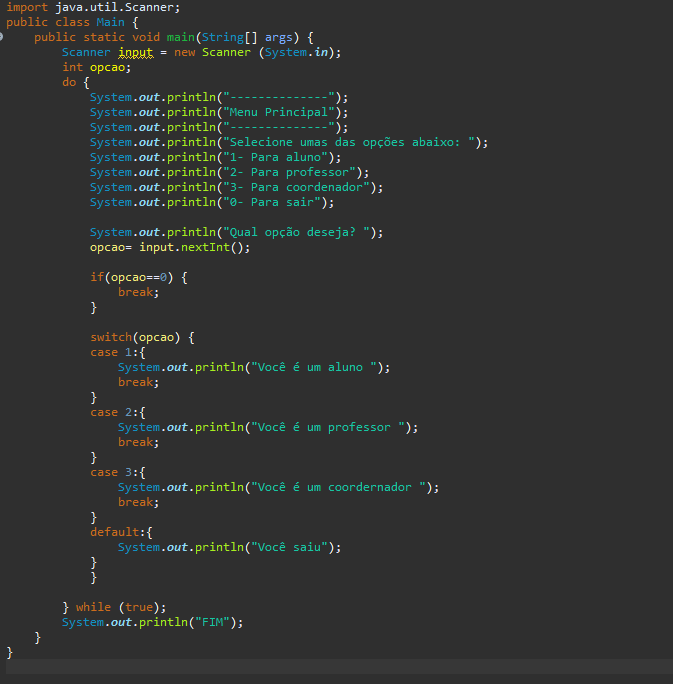
Ele **executa pelo menos 1 vez**!

Estrutura **do{...} while (condição)**



Normalmente usado quando não sei a quantidade de vezes vai usar!

**Exercício: fazer um menu no qual a pessoa escolhe as opções**



* **For:**

Usado geralmente **quando se sabe a quantidade de vezes que se vai utilizar**.

Estrutura: **for(inicialização; condição ; incremento/decremento){...}**

Texto

Descrição gerada automaticamente Uma imagem contendo Texto

Descrição gerada automaticamente

Aplicativo

Descrição gerada automaticamente com confiança baixa

Também pode ser feito a declaração fora:

Texto, Carta

Descrição gerada automaticamente

**Exercício: Calcular a media de 3 alunos.**

Texto

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

**Alguns exemplos possíveis com o for:**

1. **Inicialização de mais de uma variável colocando vírgulas entre as instruções separadas, como a seguir:**

for (g = 0**,** h = 1; g <6; ++ g)

1. **Verificação de mais de uma condição usando operadores AND ou OR, como a seguir:**

for (g = 0; g <3 && h> 1; ++ g, h–)

1. **Decrementando a variável de controle do loop e verificação de alguma outra condição, como a seguir:**

for (g = 5; g> = 1; --g)

1. **Alterando mais de um valor, como no seguinte:**

for (g = 0; g <10; ++ g, ++ h, soma + = g)

1. **Você pode deixar uma ou mais partes de um loop for vazias, embora os dois pontos e vírgulas ainda sejam necessários como espaços reservados. Por exemplo, se x foi inicializado em uma instrução de programa anterior, você pode escrever o seguinte**

for (; x <10; ++ x)

* **Loop enhanced for:**

O loop enhanced for permite percorrer um array / coleção sem especificar os pontos inicial e final da variável de controle do loop. A forma geral da versão aprimorada para loop é mostrada aqui:

**for (typeitr-var: collection) bloco de instruções**

Aqui, type especifica o tipo e itr-var especifica o nome de uma variável de iteração que receberá os elementos de uma coleção, uma de cada vez, do começo ao fim. Como a variável de interação recebe valores da coleção, o tipo deve ser o mesmo que (ou compatível com) os elementos armazenados na coleção. Portanto, ao iterar sobre matrizes, o tipo deve ser compatível com o tipo base da matriz.

EX:

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

* **Instruções de ramificação Java:**

Java fornece três instruções de ramificação: **break, continue e return**. O Break e continue em Java são duas palavras-chave essenciais que os iniciantes precisam conhecer enquanto usam loops (loop for, loop while loop do while). A instrução **break em Java é usada para interromper o loop** e transfere o controle para a linha imediatamente fora do loop, enquanto **continue é usado para escapar da execução atual** (iteração) e **transfere o controle de volta ao início do loop**. Tanto break quanto a continue permitem ao programador criar construções sofisticadas de algoritmo e loop.

* **Instruções rotuladas:**

Embora muitas instruções possam ser rotuladas, é mais comum usar rótulos com instruções de loop como for ou while, em conjunto com instruções de break e continue. Uma instrução rotulada consiste em um identificador válido que termina com dois pontos (:).

Você precisa entender a diferença entre break e continue rotulado e não rotulado. Os rotulados são necessários apenas nas situações em que você possui um loop aninhado e precisam indicar de quais loops aninhados você deseja interromper ou de quais loops aninhados deseja continuar na próxima iteração. Uma instrução break sairá do loop rotulado, em oposição ao loop mais interno, se a palavra-chave break for combinada com um rótulo.

* **A instrução return:**

A última instrução de controle é return. **Return é usada para retornar explicitamente de um método**. Ou seja, faz com que o controle do programa seja transferido de volta ao chamador do método. A qualquer momento em um método, a instrução return pode ser usada para fazer com que a execução volte ao chamador do método. Assim, return finaliza imediatamente o método em que é executada.