Comandos de repetição

Apresentação

Durante o desenvolvimento de uma solução computacional, são necessárias diversas linhas de código, formadas por estruturas de controle, variáveis, entre outras. Cada uma dessas linhas é uma peça essencial para a construção de uma aplicação.

Um tipo muito especial de estrutura de controle são os comandos de repetição, que são responsáveis por proporcionar ao desenvolvedor a possibilidade de repetir a execução de um trecho de código sem a necessidade de reescrevê-lo. Por exemplo, quando é necessário verificar quais alunos foram aprovados em uma disciplina, é improdutivo reescrever o mesmo teste para cada aluno da turma, sendo que as instruções são as mesmas. Assim, uma solução mais prática é escrever as instruções necessárias para um aluno e repetir o mesmo bloco de instruções para cada aluno da turma. Essa característica proporciona mais otimização e legibilidade ao código da aplicação.

Nesta Unidade de Aprendizagem, você vai conhecer os conceitos relacionados a comandos de repetição, sintaxe e situações de utilização. Além disso, vai conferir exemplos de como aplicá-los na resolução de problemas computacionais.

Bons estudos.

Ao final desta Unidade de Aprendizagem, você deve apresentar os seguintes aprendizados:

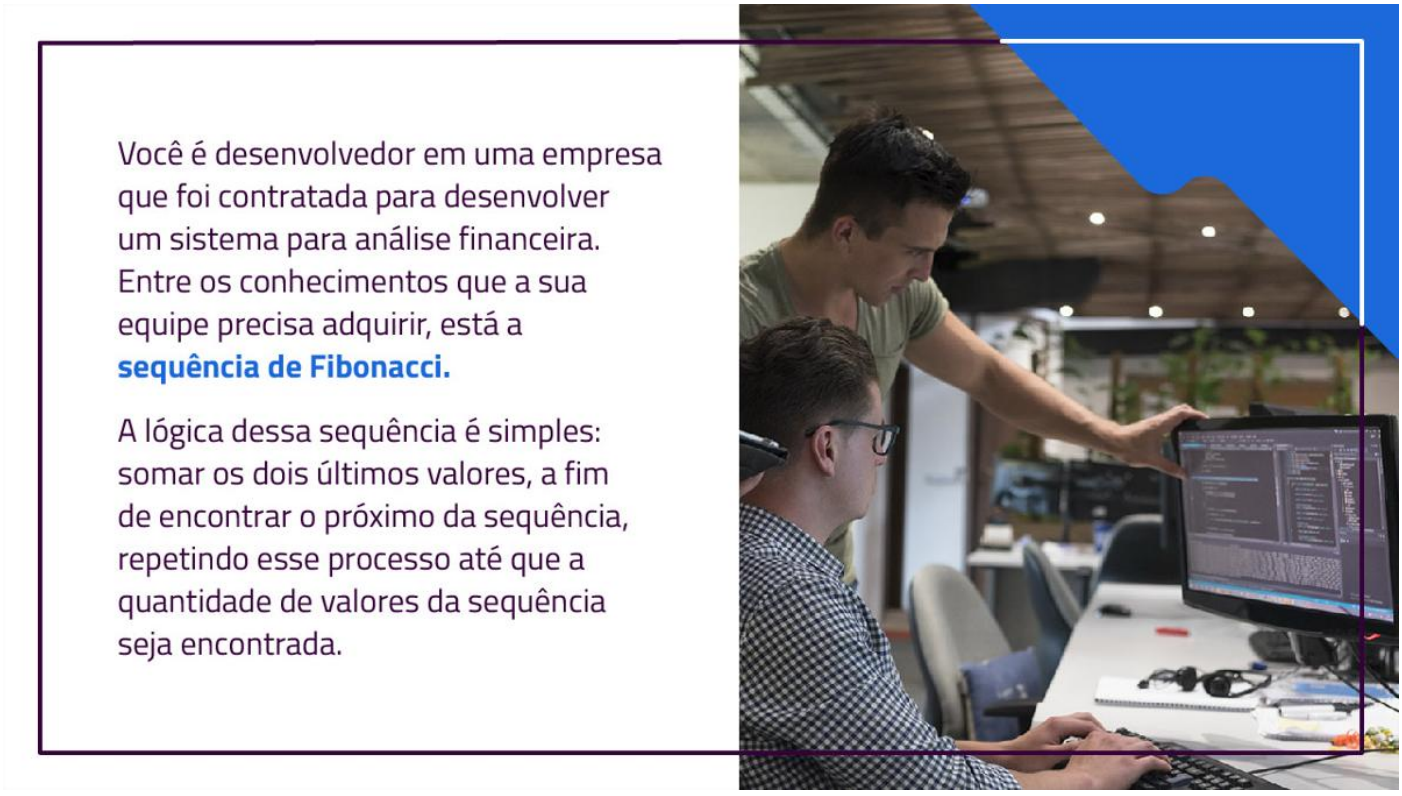
- Conceituar estrutura de repetição no contexto da programação.
- Descrever as estruturas de repetição for, while e do-while.
- Ilustrar a diferença entre as estruturas de repetição na prática.

Desafio

A matemática tem diversos números e sequências clássicas que são utilizadas em sistemas computacionais, como a sequência de Fibonacci. Trata-se de uma sequência de números inteiros, com os primeiros números sendo 0 e 1, e cada termo subsequente é definido pela soma dos dois números anteriores. Sendo assim, os primeiros dez números dessa sequência seriam 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, e assim sucessivamente.

Muito utilizada na arquitetura e nas artes, como, por exemplo, nas pirâmides, em quadros (p. ex., *Monalisa*) e em peças publicitárias, atualmente a sequência de Fibonacci é bastante implementada em aplicações de análise de mercados financeiros, estatística e teoria dos jogos. A sua formulação matemática simplificada indica uma função *g* tal que $g(n + 2) = g(n) + g(n + 1)$.

Considere a situação a seguir.



Considerando as características da sequência de Fibonacci e a necessidade de utilizar um comando de repetição, apresente um algoritmo escrito em Java que contenha os primeiros 50 números dessa sequência.

Infográfico

Do ponto de vista computacional, um algoritmo é uma sequência finita de ações para a resolução de um determinado tipo de problema; ou seja, são os passos descritos para se realizar uma tarefa. Para desenvolvê-lo, são utilizados diversos recursos, como as repetições de trechos de código.

Para tanto, a linguagem Java dispõe de três comandos para controle do fluxo de repetição: for, while e do-while. Apesar de os três comandos terem a mesma finalidade, cada um tem particularidades que se adaptam mais adequadamente a determinados tipos de problemas computacionais.

Neste infográfico, confira um resumo desses comandos, bem como o comportamento padrão de cada um deles.

```
public class FibonacciSequence {
    public static void main(String[] args) {
        int n = 50; // Número de termos desejados na sequência

        long[] fibonacciNumbers = new long[n];
        fibonacciNumbers[0] = 0; // Primeiro número
        fibonacciNumbers[1] = 1; // Segundo número

        // Use um loop for para calcular os números restantes
        for (int i = 2; i < n; i++) {
            fibonacciNumbers[i] = fibonacciNumbers[i - 1] + fibonacciNumbers[i - 2];
        }

        // Imprima os primeiros 50 números da sequência de Fibonacci
        for (int i = 0; i < n; i++) {
            System.out.print(fibonacciNumbers[i] + " ");
        }
    }
}
```

COMANDOS DE REPETIÇÃO FOR, WHILE E DO-WHILE

Os comandos de repetição são comuns em soluções algorítmicas e, por vezes, necessitam que seu comportamento nativo seja alterado.

A seguir, acompanhe um resumo dos comandos **for**, **while** e **do-while.**

Comando for

O comando for permite realizar repetições de código em tempo de execução. Contudo, para utilizá-lo, faz-se necessário saber exatamente quantas vezes o trecho deverá ser repetido. Em geral, ele é utilizado para acessar itens de algum objeto iterável, como, por exemplo, uma lista.

```
1. public class MyClass {
2.         public static void main(String args[]) {
3.         for(int i=0; i<5; i++)
4.         System.out.println("Ciclo: "+ i);
5.         }
6. }
```

No exemplo, há uma sequência de cinco ciclos. A cada ciclo, é impressa uma mensagem indicando o valor atual da variável i. Seu resultado esperado será:

```
Ciclo: 0
Ciclo: 1
Ciclo: 2
Ciclo: 3
Ciclo: 4
```

Comandos while e do-while

Os comando de repetição do tipo "**enquanto**" permitem que um bloco de instruções seja repetido uma quantidade indefinida de vezes, desde que a sua condição continue verdadeira.

Esse bloco é representado pelos comando while e do-while, que se diferenciam somente np momento que o teste de parada é realizado.

```
1. public class MyClass {
2.        public static void main(String args[]) {
3.                int contador=0;
4.                while(contador<5){
5.                System.out.println("iteração do while: "+ contador);
6.                contador++;
7.                }
8.        }
9. }
```

Importante!

Observe que o teste condicional do comando while é **realizado no topo** do comando, isso quer dizer que, se o valor da variável contador for maior que 4, o laço não será executado.

No exemplo abaixo de uso do comando while, quando o contador chegar a 5, o teste condicional retornará **falso**, forçando o laço a retornar ao início.

```
iteração do while: 0
iteração do while: 1
iteração do while: 2
iteração do while: 3
iteração do while: 4
```

No exemplo a seguir de uso do comando do-while, mesmo sabendo que o valor da variável contador retornará falso no teste de parada, o laço ainda é executado.

```
1. public class MyClass {
2.     public static void main(String args[]) {
3.         int contador=10;
4.         do{
5.            System.out.println("valor do contador: "+ contador);
6.            contador++;
7.         }while(contador<5);
8.     }
9. }
```

```
valor do contador: 10
```

Importante!

Observe que o teste condicional do comando do-while é **realizado no final**, proporcionando que as suas instruções internas sejam executadas ao menos uma vez.

Como pôde-se perceber com esses exemplos, a aplicação de comandos de repetição é muito simples, cabendo ao desenvolvedor decidir quando deve utilizá-las. Contudo, a sua aplicação pode reduzir muitas linhas de código.

Aponte a câmera para o código e acesse o link do conteúdo ou clique no código para acessar.

Conteúdo do Livro

Os comandos de repetição, também conhecidos como laços ou *loops*, são algumas das estruturas de controle de fluxo mais utilizadas no desenvolvimento de algoritmos computacionais. Por meio do seu uso, os desenvolvedores podem aplicar a lógica para a solução de um problema de conjunto de dados sem a necessidade de reescrever o mesmo conjunto de instruções para cada um dos itens do conjunto, o que seria uma tarefa morosa e improdutiva.

Em suma, há três comandos de repetição: for, while e do-while. Cada um deles tem suas próprias particularidades, sintaxe e situações de implementação mais adequadas.

No capítulo Comandos de repetição, base teórica desta Unidade de Aprendizagem, você vai conhecer os comandos de repetição, importantes estruturas de controle de fluxo de uma aplicação. Além disso, você vai conhecer a sintaxe de cada comando. Por fim, você vai conferir exemplos práticos direcionados às particularidades de cada comando.

Boa leitura.

PROGRAMAÇÃO ORIENTADA A OBJETOS

SAGAH SOLUÇÕES EDUCACIONAIS INTEGRADAS

Comandos de repetição

Marcelo da Silva dos Santos

OBJETIVOS DE APRENDIZAGEM

- Conceituar estrutura de repetição no contexto da programação.
- Descrever as estruturas de repetição `for`, `while` e `do-while`.
- Ilustrar a diferença entre as estruturas de repetição na prática.

Introdução

Os comandos de repetição são amplamente utilizados na solução de problemas computacionais. Seja em algoritmos de ordenação de dados, em uma busca em banco de dados, na criação de um *menu* de opções em um sistema ou outras situações em que precisamos repetir um mesmo comando diversas vezes até que algum objetivo esteja concluído, vamos utilizar um ou mais comandos de repetição.

As situações de uso são as mais diversas, sendo que podemos tanto repetir uma ação um número exato de vezes (por exemplo, bloquear um acesso depois de três tentativas), quanto repetir determinadas instruções até uma condição ser atendida (estudar até entender a lição, independentemente do tempo que levar). Qualquer uma das formas exige uma condição para que a repetição cesse, seja por terminar o número de iterações especificado ou por chegar ao resultado buscado.

Neste capítulo, você vai conhecer os comandos de repetição e sua importância no contexto computacional. Por meio de exemplos, você conhecerá os componentes de sua sintaxe, bem como os cenários de uso mais apropriados às características de cada comando.

Introdução aos comandos e tipos de repetição

Um *software* pode ser compreendido como um conjunto de instruções que podem ser executadas por um processador e que manipulam diversos tipos de dados (SOMMERVILLE, 2011). Em seu desenvolvimento, muitas vezes são necessárias milhares de linhas de código, dependendo da escala da aplicação. Essas linhas traduzem a lógica de programação em linguagem compreensível à máquina e, às vezes, devido à necessidade da operação, será necessário executar mais de uma vez o mesmo comando ou um conjunto de comandos, de acordo com uma condição (MANZANO; OLIVEIRA, 2010).

Esse conjunto de comandos são estruturas que têm a função de executar repetidamente uma ou mais instruções, até que uma condição pré-estabelecida indique que não seja mais necessária a execução desse bloco de instruções. Também conhecidos como laços ou *loops*, essas estruturas de controle de fluxo proporcionam ao desenvolvedor uma forma de aplicar a mesma ação diversas vezes, sem a necessidade de reescrever as instruções novamente. Para que possamos replicar o mesmo bloco de instruções *n* vezes, será indispensável que a condição continue favorável, sendo necessário testar tal condição antes do início do bloco de instruções ou após a sua execução.

Basicamente, existem dois tipos de problemas que envolvem repetições de ações: problemas com repetições limitadas e problemas com quantidade de repetições indefinidas. Apesar de as duas situações terem a mesma finalidade, a repetição de um conjunto de ações (ou instruções) de cada um tem particularidades que serão melhor empregadas dependendo do tipo de problema computacional. A seguir, essas características serão mais bem explicitadas.

Problemas com repetições limitadas

Basicamente, são problemas que envolvem um conjunto finito de repetições até que consigamos atingir o objetivo principal (HORSTMANN, 2009). Imagine que você tenha uma caixa com 10 baterias pequenas, para objetos domésticos, e, por engano, você misturou na mesma caixa algumas baterias que deveriam ser descartadas por não terem mais nenhuma carga. Como todas são iguais e você não sabe quantas baterias estão descarregadas, a única forma de separá-las é testando uma a uma. Nesse cenário, nossas instruções são: tirar uma bateria da caixa, testá-la em um aparelho e, caso tenha carga, acondicioná-la

em uma caixa rotulada; caso não tenha carga, deve ser colocada em uma embalagem para descarte. Esse conjunto de instruções deverá ser repetido dez vezes, já que este é o número de baterias.

Computacionalmente, são vários os exemplos de aplicações com número de repetições pré-definido, como calcular a nota final dos alunos de uma turma, cujo cálculo deve ser repetido para cada aluno da turma, ou contar a quantidade de clientes com a mensalidade atrasada, em que, para cada cliente, deve ser verificado o *status* da mensalidade.

Resumindo, para termos esse tipo de repetição, é necessário ter problemas computacionais que envolvem tarefas repetitivas, mas cuja quantidade de repetições necessárias para atingirmos a resolução do problema é conhecida previamente. O comando de repetição que tem essas particularidades é o comando `FOR`, muitas vezes traduzido como "para cada/faça".

Problemas com repetições indefinidas

Ao contrário do cenário anterior, aqui teremos problemas computacionais em que não sabemos ao certo quantas vezes devemos repetir o conjunto de ações (HORSTMANN, 2009). Imagine uma situação semelhante à anterior, uma caixa com dez baterias, em que não sabemos a quantidade de baterias novas, nem quais são. Dessa vez, precisamos substituir as duas baterias de um controle remoto, portanto não precisamos testar todas, apenas testar até identificar duas baterias com carga.

Nesse exemplo, o que importa não é quantas baterias vamos testar, mas o resultado; ou seja, caso as duas primeiras baterias testadas sejam utilizáveis, podemos parar de procurar, pois nosso objetivo foi atingido. É claro que pode ocorrer de termos que testar até a última bateria para chegar ao resultado.

Computacionalmente, qualquer exemplo que siga a necessidade de repetir um mesmo conjunto de ações enquanto não atingimos o objetivo pode ser implementado com essa abordagem. Por exemplo, um jogo de adivinhação no qual o usuário deve tentar acertar o número escolhido pelo computador; nesse caso, também não conseguimos prever quantas vezes o usuário vai "chutar" até que consiga acertar. Ou, então, a validação de valores de entrada em um formulário; nesse caso, sempre que o usuário digitar um valor fora do intervalo esperado, uma mensagem solicitando que ele corrija sua digitação deve ser disparada enquanto ele não digitar um valor válido (HORSTMANN, 2009).

Esse tipo de repetição é representado por dois comandos diferentes: o `while` e o `do-while`. Note que a palavra `while`, traduzida para a língua portuguesa como "enquanto", é um termo que define muito bem esse tipo de problema. A explicação para termos dois comandos para a mesma categoria é devido às características do problema e ao momento em que a verificação de parada é realizada (HORSTMANN, 2009). A diferença básica é que o comando `while` verifica seu critério de parada antes de realizar o bloco de instruções, ao contrário do comando `do-while`, que primeiro executa o bloco de instruções para depois realizar o teste.

Pelo que você percebeu, os comandos de repetição são basicamente três: `for`, `while` e `do-while`. Na próxima seção, você conhecerá a sintaxe de cada comando na linguagem Java.

Sintaxe dos comandos de repetição

Como foi apresentado na seção anterior, os comandos de repetição, também conhecidos como estruturas de repetição, laços ou *loopings*, permitem que um trecho de código seja executado repetidamente enquanto uma determinada condição for verdadeira (HORSTMANN, 2009). Também foi visto que as estruturas de repetição necessitam de três parâmetros básicos para seu funcionamento, sendo eles:

- um ponto de partida — variável que indica o valor inicial para contagem ou o valor inicial de referência;
- um critério de parada — um teste condicional, que é executado a cada ciclo comparando se atingimos nosso objetivo, sendo que indicará a continuidade da repetição sempre que seu resultado retorne TRUE (verdadeiro);
- atualização da variável de referência — ao final de cada iteração, o valor da variável de referência deve ser atualizado, seja com um incremento/decremento (`for`) ou outra atualização conforme o problema (`while` ou `do-while`).

Cada uma das três formas de implementar um comando de repetição organiza esses parâmetros conforme as definições apresentadas na seção anterior. A seguir, você vai conhecer a estrutura de cada um dos comandos de repetição.

Comando de repetição `for`

O comando `for` é implementado em casos em que o bloco de instruções deva ser executado um determinado número de vezes. Sua sintaxe incorpora os três parâmetros em sua sintaxe básica, ou seja, os três são implementados dentro da estrutura do comando (HORSTMANN, 2007). Você pode observar sua estrutura a seguir.

```
for (<ponto de partida>; <condição de parada>; <incremento/
decremento>) {
    // BLOCO DE COMANDOS QUE DEVEM SER EXECUTADOS
}
```

Para sua construção, necessitamos da instância de uma variável para o controle do laço; para tanto, a instanciamos diretamente no corpo do comando, seu tipo deve ser int (inteiro) e seu valor inicial dependerá do problema. Você pode nomeá-la como quiser, mas saiba que, por convenção, é comum utilizar a letra *i* (de *index*) como nome.

O critério de parada é definido por uma comparação entre o valor final e a variável *i*. Assim, considerando o exemplo anterior, onde:

- *i* seja inicializado com o valor 0 (int i=0;);
- precisamos repetir 10 vezes o bloco de instruções (0, 1, 2, 3, 4, 5, 6, 7, 8 e 9, ou seja, 10 repetições);
- o bloco será repetido sempre que o teste retorne verdadeiro, logo, i < 10 só será falso quando o valor de *i* chegar a 10;
- incrementos e decrementos são funções comuns em Java. Para aumentarmos em um o valor de *i*, basta adicionar duas vezes o sinal de "+" ao final da variável, dessa forma i++. Da mesma forma, para decrementar, utilizamos a sintaxe i--.

Substituindo os parâmetros na estrutura do comando, teremos seu formato completo. Observe sua estrutura completa.

```
for(int i=0; i < 10; i++){
    // BLOCO DE COMANDOS QUE DEVEM SER EXECUTADOS
}
```

A primeira expressão é executada apenas uma vez, no início do laço. A condição de parada é, então, avaliada para verificar se o laço deve terminar. Caso a expressão seja verdadeira (isto é, menor ou igual a 9), o corpo da repetição é executado. Depois dessa execução, a expressão de incremento é executada e o processo é repetido a partir da condição de parada. O bloco de comandos, por sua vez, pode ser composto por uma ou muitas instruções.

Comando de repetição `while`

Como anteriormente mencionado, o comando `while` é mais bem empregado em repetições do tipo "até que algo aconteça". Apesar de sua sintaxe também utilizar os três parâmetros, nem todos estão incorporados à estrutura do comando (HORSTMANN, 2007). Você pode observar sua estrutura a seguir.

```
while (condição de parada){
    // BLOCO DE COMANDOS QUE DEVEM SER EXECUTADOS enquanto
o teste condicional for igual a verdadeiro (true)
    // a atualização da condição de parada deve ser reali-
zada neste local
}
```

Note que a condição de parada é declarada no início do comando, antes do bloco de comandos. Primeiramente é realizado o teste da condição de parada. Como o teste está no início da estrutura, caso o teste resulte em falso, o bloco de comandos não será executado.

Outro ponto a ser destacado é que, uma vez iniciado o laço, ele somente será encerrado quando o teste for falso; logo, se não incluirmos no bloco de comandos, a atualização do valor da variável de referência nunca será alterada e o laço será infinito, ou seja, sem ponto de saída. É comum, no desenvolvimento, criar equivocadamente *loops* infinitos, em que a expressão permanece verdadeira para sempre.

Comando de repetição `do-while`

Similar ao `while`, sua principal diferença em relação ao anterior é o momento em que a condição de parada é verificada, nesse caso, ao final da execução do bloco de instruções, ou seja, as instruções dentro do laço serão executadas pelo menos uma vez (HORSTMANN, 2007).

```
do {
    // BLOCO DE COMANDOS QUE DEVEM SER EXECUTADOS enquanto
o teste condicional for igual a verdadeiro (true)
    // a atualização da condição de parada deve ser reali-
zada neste local
} while (condição de parada);
```

Estruturalmente, a única mudança na sintaxe do comando é a posição do teste, embora essa mudança altere consideravelmente o resultado de sua operação, considerando que seu bloco de comando será executado ao menos a primeira vez. Note também que, assim como o comando `while`, o comando `do-while` também necessita que sua variável de controle seja atualizada dentro do bloco de instruções para que não ocorra a possibilidade de *looping* infinito.

Agora que você já conhece a aplicabilidade de cada estrutura e sua sintaxe básica, na próxima seção serão apresentados exemplos de utilização de cada comando para auxiliar seu aprendizado.

Exemplos de uso dos comandos de repetição

Agora que você já conhece os comandos de repetição e suas particularidades e o melhor uso para cada um deles, serão apresentados alguns exemplos de problemas computacionais simples que podem ser resolvidos aplicando repetições.

Vamos iniciar por problemas com contagem pré-definida e, em seguida, passamos a problemas com quantidade de repetições indefinida, tanto aquelas que precisam executar o bloco de instruções ao menos uma vez quanto aquelas que só devem executar se o teste condicional assim indicar.

Exemplo de implementação do comando `for`

Essa categoria de comandos de repetição executará o bloco de instruções utilizando dois parâmetros básicos, um valor de início e um valor final. Normalmente, esses valores são numéricos e indicam quantas vezes o bloco será repetido (HORSTMANN, 2009). Esses valores são utilizados para facilitar o controle da contagem das repetições, também chamado de ciclos ou passos.

Por exemplo, para o caso do teste das baterias comentado na primeira seção, caso a nossa contagem seja inicializada com o valor zero, devemos prosseguir até contar dez passos, nesse caso parando assim que chegarmos a nove (0, 1, 2, 3, 4, 5, 6, 7, 8 e 9, ou seja, 10 repetições). Para facilitar essa contagem, seu valor a cada ciclo é armazenado em uma variável, e, como a contagem é feita uma a uma, a cada ciclo deve ser somado o valor 1 a essa variável contadora. Essa ação de "somar 1" ao valor de uma variável é chamada de incremento. Caso a contagem seja regressiva (iniciando em 9 e terminando em 0), a ação de "subtrair 1" é chamada de decremento.

Como pode-se perceber, são problemas que envolvem, de maneira geral, uma contagem. Para isso, podemos utilizar a variável *i* como auxiliar para essa tarefa; como seu valor é alterado a cada ciclo, ela guarda o índice do passo atual. Vamos iniciar com um exemplo simples, mostrar os números de 1 a 9 em sequência.

Para tanto, vamos configurar o controle do número de repetições. Como o primeiro número a ser impresso é o 1, devemos inicializar a variável *i* com esse valor. O último número impresso será 9 e, sabendo que para o laço ser interrompido, o teste condicionado deve retornar FALSO, nosso teste condicional deve permitir que o laço seja executado para todo *i* que for menor que 10, assim i < 10. Para finalizar, incluímos um incrementador i++.

```
for(int i=1; i<10;i++){
    System.out.println(i);
}
```

Também poderíamos imprimir de forma regressiva, iniciando em 9 e terminando em 1. Nesse caso, a inicialização de *i* será com o valor 9, o critério de parada passa a ser i > 0 (quando *i* for igual a 0, o teste retornará falso e o laço encerrará) e agora devemos decrementar o contador *i*, pois se não subtrairmos a cada ciclo, nunca chegaremos ao 0 e entraremos em um *looping* infinito (não conseguirá parar). A implementação do exemplo pode ser vista a seguir.

```
for(int i=9; i>0;i--){
    System.out.println(i);
}
```

Agora, em outro exemplo, você precisa que o algoritmo indique todos os valores pares no intervalo entre 150 e 200, incluindo esses valores. O teste para verificação de números pares é implementado com o comando `if` (se), mas utilizaremos o nosso índice como parte da solução, já que devemos realizar o teste de verificação para cada número do intervalo. Observe a resolução a seguir.

```
for(int i=150; i <= 200 ;i++){
    if(i%2 == 0){
        System.out.println(i);
    }
}
```

Nesse exemplo, o bloco de instruções contém mais comandos, mas note que o contador foi mais uma vez utilizado como parte da solução. Basta configurar corretamente os parâmetros, ou seja, a configuração dos parâmetros do `for` faz parte da solução do problema.

Exemplo de implementação do comando `while`

Como não necessitamos de contagem sequencial, lembrando que o comando `while` é adequado para problemas "enquanto algo ocorrer", os problemas implementados com estruturas `while` (e `do-while` inclusive, conforme veremos em breve) não necessariamente serão numéricos. Por exemplo, no caso de precisarmos de uma confirmação do tipo "deseja continuar?", comum em jogos, podemos inicializar nossa variável de controle (note que mudamos o nome, pois não precisamos contar o número de passos) com o valor "S" (representando a opção "sim"); logo, enquanto a opção do jogador para a questão "deseja continuar?" for o valor "S" (ou seja, enquanto a condição for verdadeira), o jogo será reiniciado.

Resumindo, para termos esse tipo de repetição, necessitamos de uma variável com nosso valor de referência (variável que guarda o "S") para comparação, uma condição de parada (enquanto o usuário digitar "S") e uma forma de atualizar o valor da variável de referência (pergunta "deseja continuar?").

Como exemplo inicial, podemos realizar uma validação de números. Digamos que você necessita implementar uma calculadora; nesse caso, o sistema deve solicitar que você digite os valores a serem calculados e a operação.

Os cálculos são simples, mas, ao executarmos uma divisão, necessitamos incluir uma validação; considerando que estamos solicitando que os números sejam informados pelo usuário, estamos possibilitando que ele informe qualquer número, inclusive o valor 0. Considerando que, pelas regras básicas da matemática, é impossível a divisão por 0, devemos garantir que o usuário não digite 0 para o segundo valor; nesse caso, devemos solicitar que ele informe um valor diferente.

Esse exemplo se encaixa no uso de `while`, considerando que, no caso de o usuário informar um valor válido, não será necessário executar o bloco de comando do `while`. Já no caso de o valor digitado ser 0 (variável de referência), o fluxo deverá executar o bloco de instruções que conterá uma mensagem de erro, a solicitação de nova digitação ea atualização da variável de referência (para que possamos sair do ciclo). Assim que encerrarem as ações do bloco de instruções, retornamos para o teste condicional, no qual será novamente verificado se o valor continua inválido. Sabendo que o comando continuará repetindo enquanto o teste resultar TRUE (segundoNum == 0), somente quando o valor digitado for diferente de 0 o ciclo encerrará. Observe a implementação do exemplo a seguir.

```
Scanner scanner = new Scanner(System.in);
float primeiroNum = scanner.nextFloat();
float segundoNum = scanner.nextFloat();

while(segundoNum==0){
    System.out.println("digite um valor diferente de 0");
    segundoNum = scanner.nextFloat();
}
// demais instruções do algoritmo
```

Aqui precisamos incluir o uso de um *scanner*, biblioteca com funções para a leitura de entrada de dados pelo teclado. Veja que só necessitamos realizar a conferência do `segundoNum`, pois é nosso objeto de validação. O laço de repetição pode não ser necessário caso o valor informado esteja correto, mas, caso seja acionado, não temos como prever quantas vezes o laço será executado. Esse é o funcionamento padrão de um comando de repetição `while`.

Exemplo de implementação do comando `do-while`

Em partes semelhantes ao comando `while`, quando não podemos prever quantas vezes ele será executado, no comando `do-while` temos certeza de que ele irá executar ao menos uma vez (HORSTMANN, 2009).

Bastante utilizado em situações do tipo "deseja continuar?", podemos utilizar como exemplo a criação de um *menu* em um sistema, quando, após a execução inicial, podemos solicitar ao usuário se ele deseja encerrar a aplicação ou continuar executando outras ações. Observe o exemplo a seguir.

```
int opcao; // será inicializada durante a primeira execução
do ciclo do..while
Scanner entrada = new Scanner(System.in);
do {
    System.out.println("Opção 1 - Cadastrar ...");
    System.out.println("Opção 0 - Sair do programa");
    opcao = entrada.nextInt();
    //demais instruções foram omitidas
} while( opcao != 0 );

System.out.println("Programa finalizado.");
```

Neste caso, o *menu* deve ser apresentado já na primeira execução e ser novamente mostrado ao usuário sempre que ao final da ação ele indicar a opção 0; considerando que o ciclo deve se repetir sempre que o usuário indicar qualquer das opções gerais do *menu*, enquanto a opção selecionada for diferente de 0 (a opção de saída), o ciclo continuará, sem termos a definição de quantas vezes será necessário manter a repetição.

Como você percebeu, os comandos de repetição são utilizados nas mais diversas situações, auxiliando na resolução de problemas computacionais repetitivos, em que um mesmo bloco de instruções deve ser reutilizado. O cenário de aplicação auxilia a definir o tipo de ciclo a ser utilizado, sabendo que problemas com número de repetições previamente conhecidos são mais bem resolvidos utilizando o comando de repetição `for`; e problemas em que não conseguimos prever o número máximo de repetições são mais bem implementados pelos comandos `while` e `do-while`.

Com esse conhecimento, um desenvolvedor será capaz de resolver grande parte dos problemas computacionais, desde pequenas aplicações de cálculo até a mineração de grandes volumes de dados.

Referências

HORSTMANN, C. *Padrões e projetos orientados a objetos*. 2. ed. Porto Alegre: Bookman, 2007.

HORSTMANN, C. *Conceitos de computação com Java*: compatível com Java 5 & 6. 5. ed. Porto Alegre: Bookman, 2009.

MANZANO, J. A. N. G.; OLIVEIRA, J. F. *Algoritmos*: lógica para desenvolvimento de programação de computadores. 23. ed. São Paulo: Érica, 2010.

SOMMERVILLE, I. *Engenharia de software*. 9. ed. São Paulo: Pearson, 2011.

Leituras recomendadas

PRESSMAN, R. S. *Engenharia de software*: uma abordagem profissional. 7. ed. Porto Alegre: AMGH, 2011.

SCHILDT, H. *Java para iniciantes*: crie, compile e execute programas Java rapidamente. 6. ed. Porto Alegre: Bookman, 2015.

Conteúdo:

SAGAH
SOLUÇÕES EDUCACIONAIS INTEGRADAS

Dica do Professor

As estruturas de repetição são utilizadas em uma ampla variedade de problemas computacionais, desde o processamento de longas listas de itens até a construção de simples menus de aplicações. Também conhecidas como laços ou ciclos de repetição, essas estruturas são utilizadas para executar repetidamente uma ou mais instruções enquanto a sua condição de parada estiver sendo satisfeita.

Cada ciclo, ou iteração, sempre executará todo o conteúdo do bloco de instruções. Para ampliar as possibilidades, pode-se utilizar as estruturas de repetição em conjunto com outras estruturas de controle, como os comandos break e continue, capazes de alterar o comportamento padrão de um comando de repetição.

Nesta Dica do Professor, confira os comandos break e continue e acompanhe alguns exemplos de uso.

Aponte a câmera para o código e acesse o link do conteúdo ou clique no código para acessar.

Exercícios

1) **Os comandos de repetição while e do-while geralmente causam dúvidas em um primeiro contato, o que se torna trivial conforme o desenvolvedor ganha experiência.**

Considerando as diferenças entre os comandos de repetição while e do-while, assinale a alternativa correta.

A) Enquanto o comando do-while verifica a sua parada no início, o while a verifica no fim.

B) O comando while executará enquanto verdadeiro, ao passo que o comando do-while executará enquanto falso.

C) O comando while irá executar enquanto falso, ao passo que o do-while executará enquanto verdadeiro.

D) O comando while executa com número de repetições predefinido, ao contrário do comando do-while.

E) O comando while verifica a sua condição de parada no início, ao passo que o comando do-while a verifica no fim.

2) **Dependendo do tipo de problema computacional a ser implementado, por vezes, faz-se necessário utilizar comandos auxiliares para a sua execução. Assinale a alternativa correta em relação ao funcionamento do comando break.**

A) Pula a execução atual, seguindo diretamente para a próxima execução.

B) Define uma forma de depuração para avaliar comandos de repetição.

C) Interrompe permanentemente a execução de um comando de repetição.

D) Define que repetições sem critério de parada não serão executadas.

E) Interrompe a execução de uma repetição após a primeira execução.

3) **Os comandos de repetição podem desviar o fluxo das instruções durante a execução de um algoritmo repetindo trechos específicos, caso uma condição seja atendida. Analise as afirmações a seguir e classifique-as em verdadeiras (V) ou falsas (F).**

( ) O controle do comando de repetição do tipo for é realizado por meio de uma variável lógica, que é iniciada como verdadeira, interrompendo as repetições quando seu valor for modificado para falso.
( ) No comando de repetição do tipo while, o critério de parada é verificado por meio de um teste lógico no início do laço; se o resultado for falso logo na primeira execução, as suas instruções não serão executadas.
( ) Somente os comandos de repetição for e do-while controlam suas repetições por meio de verificações lógicas.
( ) As instruções do comando de repetição do-while serão executadas ao menos uma vez, pois seu critério de parada é verificado somente ao final do laço.

Assinale a alternativa que preenche as lacunas de forma correta.

A) F–F–V–F.

B) F–V–F–V.

C) V–F–V–F.

D) V–F–F–V.

E) F–V–V–F.

4) **Os comandos de repetição são comumente utilizados para realizar tarefas longas e cansativas, como somatórios ou produtos em sequências. Considerando a necessidade de se realizar o somatório de uma sequência de 100 números variáveis, com a soma inicializada com o valor 0, assinale a alternativa correta.**

A) for (int i = 1; i < 100; i++){soma = soma + i}

B) for (int i = 0; i < 99; i++){soma = soma + i}

C) for (int i = 1; i < 99; i++){soma = soma + i}

D) for (int i = 0; i <= 100; i++){soma = soma + i}

E) for (int i = 1; i < 101; ++i){soma = soma + i}

5) **Os comandos de repetição, também conhecidos como laços, são utilizados para a repetição de trechos de instruções que podem ser executados mais de uma vez, dependendo do problema implementado. Analise as afirmações a seguir e classifique-as em verdadeiras (V) ou falsas (F).**

( ) É possível realizar um comando de repetição com critério de parada no fim do laço de repetição.
( ) Se o resultado do teste do critério de parada for falso, a execução do algoritmo permanecerá no laço.
( ) O comando de repetição deve encerrar quando o teste do critério de parada for verdadeiro.
( ) Se o resultado do teste do critério de parada for verdadeiro, a execução do algoritmo permanecerá no laço.
( ) É possível realizar um comando de repetição com critério de parada no início do laço de repetição.

Assinale a alternativa que preenche as lacunas de forma correta.

A) V–F–F–V–V.

B) V–F–F–V–F.

C) F–V–F–V–F.

D) V–V–F–F–V.

E) F–F–V–V–F.

Na prática

Em diversas áreas do conhecimento que utilizam soluções computacionais, como indústria, negócios, gestão governamental, educação, entre outras, existem problemas de ordenação, tanto para registros em bancos de dados como para a manipulação de informações em arquivos, por exemplo.

Como forma de solucionar esses problemas, com frequência são utilizados algoritmos clássicos de ordenação, cujo estudo é importante para a experiência de um desenvolvedor. Ao conhecer esses algoritmos, o desenvolvedor aprimorará seus conhecimentos em lógica e seus principais recursos, como as estruturas de repetição e condicionais. Em geral, são estudados primeiro os algoritmos clássicos, como o algoritmo da bolha, base para diversas soluções computacionais.

Neste Na Prática, confira a aplicação dos comandos de repetição para ordenação de dados em um vetor.

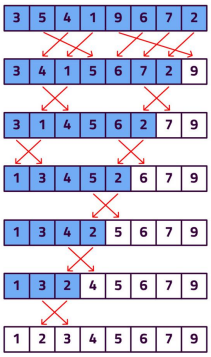
IMPLEMENTANDO UM ALGORITMO DE ORDENAÇÃO

Os algoritmos de ordenação são amplamente utilizados na área de computação, como, por exemplo, para ordenar valores (ordem crescente/decrescente), colocar os nomes em ordem alfabética, entre outras aplicações.

Mariane está iniciando seu estágio em uma equipe de desenvolvimento e está trabalhando em um problema de ordenação.

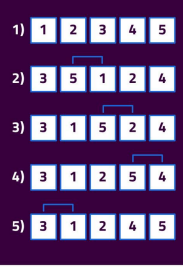
Mariane recebeu a tarefa de desenvolver uma função para ordenação de valores inseridos por meio de importação de arquivos.

Como ainda não havia trabalhado com problemas de ordenação, Mariane optou por iniciar seus estudos com um dos algoritmos mais conhecidos, o algoritmo bubble sort, ou algoritmo da bolha.



O algoritmo utiliza múltiplas repetições, em que, a cada ciclo, são comparados os elementos adjacentes (dois a dois) de um vetor; por exemplo, comparando o primeiro número com o segundo, o segundo com o terceiro, e assim sucessivamente.

Nessa comparação, se o primeiro valor for menor que o segundo, ambos trocam de posição. Dessa forma, os valores menores são movimentados em direção à extremidade do vetor, imitando o movimento de uma bolha de ar.



Desse modo, Mariane utilizou um pequeno vetor de números inteiros para seus testes. Para compreender como o tamanho do vetor é conhecido, ela utilizou um comando for configurado para repetir a comparação e a possível troca a cada ciclo.

```
public class MyClass {
    public static void main(String args[]) {

        int[] vetor = {4, 9, 8, 3, 2};
        int aux = 0;
        int i = 0;

        System.out.println("Lista de valores desordenados: ");
        for(i = 0; i<5; i++){
                System.out.println(" "+vetor[i]);
        }

        for(i = 0; i<5; i++){
                for(int j = 0; j<4; j++){
                        if(vetor[j] > vetor[j + 1]){
                                aux = vetor[j];
                                vetor[j] = vetor[j+1];
                                vetor[j+1] = aux;
                        }
                }
        }
        System.out.println("\nLista de valores ordenados:");
        for(i = 0; i<5; i++){
                System.out.println(" "+vetor[i]);
        }

    }
```

Como o processo é recursivo, se as repetições não forem suficientes para concluir a ordenação, todo o for deve ser repetido novamente, necessitando de duas estruturas de repetição.

Após seus testes, Mariane observou que, apesar de haver algoritmos de ordenação mais eficazes, o algoritmo da bolha auxiliou muito o seu aprendizado. A partir dele, Mariane aumentou seus conhecimentos sobre algoritmos recursivos.

```
Lista de valores desordenados:
4
9
8
3
2

Lista de valores ordenados:
2
3
4
8
9
```

Atualmente, Mariane sabe utilizar os recursos para ordenação de dados fornecidos pela linguagem Java, porém seu estudo anterior proporcionou um grande aprendizado quanto ao uso de comandos de repetição.

Aponte a câmera para o código e acesse o link do conteúdo ou clique no código para acessar.

Saiba mais

Para ampliar o seu conhecimento a respeito desse assunto, veja a seguir as sugestões do professor:

Programar em Java – ciclo/laço FOR

Acesse o vídeo a seguir para saber mais sobre o comando de repetição for e sua implementação na linguagem Java.

Aponte a câmera para o código e acesse o link do conteúdo ou clique no código para acessar.

Programar em Java – DO-WHILE (laço/ciclo/*loop*)

Assista ao vídeo a seguir para saber mais sobre o comando de repetição DO-WHILE e sua implementação na linguagem Java.

Aponte a câmera para o código e acesse o link do conteúdo ou clique no código para acessar.

Java para iniciantes: crie, compile e execute programas Java rapidamente

Leia o livro a seguir para saber mais sobre os comandos de repetição e seu relacionamento com as demais estruturas de controle utilizadas na linguagem Java.

Conteúdo interativo disponível na plataforma de ensino!

Programar em Java – WHILE (laço/ciclo/*loop*)

Confira o vídeo a seguir para saber mais sobre o comando de repetição WHILE e sua implementação na linguagem Java.

Aponte a câmera para o código e acesse o link do conteúdo ou clique no código para acessar.