ATITUS
EDUCAÇÃO

Estruturas de Repetição em Java: Dominando Loops para Tarefas Repetitivas

No mundo da programação, a repetição é crucial para automatizar tarefas tediosas e otimizar seu código. Em Java, as estruturas de repetição fornecem ferramentas poderosas para executar um bloco de código várias vezes, desde iterar sobre coleções até realizar verificações repetitivas. Neste guia, exploraremos as principais estruturas de repetição em Java, desde o clássico `for` até o versátil `do-while`, munindo você com o conhecimento necessário para dominar loops e escrever código mais eficiente.

`for each`: O Loop "For" Simplificado para Coleções

O loop `for each`, também conhecido como "loop foreach", é a maneira mais simples e elegante de iterar sobre os elementos de uma coleção em Java. Ele percorre cada elemento da coleção, um por um, e atribui-o a uma variável definida pelo usuário. Imagine que você tem uma lista de nomes de alunos e deseja exibi-los na tela. Com o `for each`, você pode fazer isso com apenas algumas linhas de código:

```
List<String> nomesAlunos = Arrays.asList("Ana", "João", "Maria");

for (String nome : nomesAlunos) {
  System.out.println(nome);
}
```

Neste exemplo, a variável `nome` recebe cada nome da lista `nomesAlunos` a cada iteração do loop. O `for each` esconde a complexidade do loop `for` tradicional, tornando-o ideal para iniciantes e para lidar com coleções.

Curiosidade: Você sabia que o loop `for each` é compilado para um loop `for` tradicional? Isso significa que, internamente, o Java usa o `for` para iterar sobre os elementos da coleção, mas oferece a sintaxe simplificada do `for each` para facilitar sua vida.

`for`: O Loop Tradicional com Controle Total

O loop `for`, também conhecido como "loop for tradicional", oferece controle granular sobre a execução do loop, permitindo definir a inicialização da variável de controle, a condição de parada e o incremento (ou decremento) a cada iteração. Ele é ideal para situações em que você precisa de mais flexibilidade e controle do que o `for each` oferece.

Estrutura do Loop `for`:

```
for (inicialização; condição; incremento) {
  // Corpo do loop
}
```

Exemplo: Imprimindo números de 1 a 10:

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

Detalhes Importantes:

- **Inicialização:** A variável de controle é inicializada com o valor especificado.
- **Condição:** O loop continua enquanto a condição for `true`. Se a condição for `false`, o loop termina.
- **Incremento (opcional):** A cada iteração, o valor da variável de controle é incrementado (ou decrementado) pelo valor especificado. Se nenhum valor for especificado, o incremento padrão é 1.
- **Corpo do loop:** O código dentro do loop será executado para cada iteração.

Variantes do Loop `for`:

- **Laço infinito:** Se a condição for sempre `true`, o loop nunca terminará. Utilize com cautela!
- **Laço sem corpo:** Se o corpo do loop estiver vazio, o bloco de código será ignorado a cada iteração. Útil para contagem ou inicialização de variáveis.
- **Múltiplas variáveis de controle:** É possível ter mais de uma variável de controle, separadas por vírgulas.
- **Condição complexa:** A condição pode ser qualquer expressão booleana complexa.
- **Incremento negativo:** Se o incremento for negativo, o loop iterará em ordem decrescente.

`while`: Verificando Primeiro, Executando Depois

O loop `while` executa um bloco de código enquanto uma condição específica for `true`. Ele é útil quando você precisa verificar uma condição antes de executar o código do loop. Imagine que você deseja ler números do teclado até que um valor específico seja digitado. O `while` é perfeito para essa tarefa:

```
        Scanner scanner = new Scanner(System.in);
        System.out.println("Digite um número (ou 0 para sair):");

        while (true) {
            if (!scanner.hasNextInt()) {
                System.out.println("Entrada inválida. Digite um número
inteiro:");
                scanner.next(); // Limpa a entrada incorreta
```

```
            continue; // Retorna ao início do loop
        }

        int numero = scanner.nextInt();

        if (numero == 0) {
            System.out.println("Saindo do programa...");
            break; // Sai do loop
        }

        System.out.println("Você digitou: " + numero);
        System.out.println("Digite outro número (ou 0 para sair):");
    }

    scanner.close(); // Fecha o Scanner
```

`do-while`: Executando Primeiro, Verificando Depois

O loop `do-while` é semelhante ao `while`, mas com uma diferença crucial: o bloco de código do loop é executado pelo menos uma vez, antes que a condição seja verificada. Isso significa que, mesmo que a condição seja `false` na primeira iteração, o código do loop será executado. Imagine que você precisa rolar um dado virtual até obter um resultado específico. O `do-while` garante que o dado seja rolado pelo menos uma vez:

```
Random random = new Random();
int resultadoDado;

do {
  resultadoDado = random.nextInt(6) + 1; // Gera um número aleatório entre
1 e 6
  System.out.println("Você tirou: " + resultadoDado);
} while (resultadoDado != 6); // Repete até tirar 6
System.out.println("Parabéns! Você tirou 6!");
```

Escolhendo a Estrutura de Repetição Ideal

A escolha da estrutura de repetição correta depende do contexto e do que você deseja alcançar. Aqui estão algumas dicas:

- **`for each`:** Use para iterar sobre coleções de forma simples e elegante.
- **`for`:** Use quando você precisa de controle preciso sobre o loop, como definir a inicialização, condição e incremento.
- **`while`:** Use quando a condição precisa ser verificada antes de cada iteração.
- **`do-while`:** Use quando o código do loop precisa ser executado pelo menos uma vez, independentemente da condição.