Estruturas de Repetição

Elas permitem que um trecho de código seja executado repetidamente até que uma condição seja atingida. Em Java, existem três tipos de estruturas de repetição: **for**, **while** e **do-while**.

Um detalhe importante que serve para todas as estruturas de repetição é o comando **continue**, que pode ser empregado para pular uma execução do laço de repetição.

For

A estrutura de repetição **for** é utilizada quando se sabe exatamente quantas vezes o trecho de código deve ser executado. Ela é composta por três partes: a inicialização, a condição e a atualização. A inicialização é executada apenas uma vez no início do laço; a condição é avaliada a cada iteração, e enquanto ela for verdadeira, o trecho de código dentro do laço será executado; a atualização é executada após cada iteração. A sintaxe do **for** é a seguinte:

```
for (inicialização; condição; atualização) {
    // trecho de código a ser repetido
}
```

Exemplificando

```
for (int i = 0; i < 10; i++) {
    System.out.println("Valor de i: " + i);
}
```

While

A estrutura de repetição **while** é utilizada quando não se sabe quantas vezes o trecho de código deve ser executado. Ela é composta apenas pela condição, que é avaliada antes de cada iteração. Enquanto a condição for verdadeira, o trecho de código dentro do laço será executado. A sintaxe do **while** é a seguinte:

```
while (condição) {
    // trecho de código a ser repetido
}
```

Exemplificando

```
int i = 0;
while (i < 10) {
    System.out.println("Valor de i: " + i);
    i++;
}
```

Do-While

A estrutura de repetição **do-while** é semelhante ao **while**, mas com uma diferença importante: o trecho de código dentro do laço é executado pelo menos uma vez, mesmo que a condição seja falsa. A sintaxe do **do-while** é a seguinte:

```
do {
    // trecho de código a ser repetido
} while (condição);
```

Exemplificando

```
int i = 0;
do {
    System.out.println("Valor de i: " + i);
    i++;
} while (i < 10);
```

Temos também dois tipos de laços extras o **foreach e o stream**, que apesar de ser bem específico, utilizado em casos de listas e arrays, é bastante utilizado no dia a dia do desenvolvimento. Sua função é passar por todos os elementos da lista, porém contém inúmeros outros recursos que podem ser utilizados facilmente durante a repetição dos elementos da lista.

Foreach

A estrutura de repetição **foreach** é utilizada para percorrer elementos de uma coleção, como um array ou uma lista. Ela é composta por uma variável de iteração e a coleção a ser percorrida. A sintaxe do **foreach** é a seguinte:

```
for (tipo variável : coleção) {
    // trecho de código a ser repetido
}
```

Exemplificando

```
int[] numeros = {1, 2, 3, 4, 5};
for (int numero : numeros) {
    System.out.println("Valor do número: " + numero);
}
```

Stream

A API de stream do Java permite realizar operações em coleções de elementos de forma funcional e encadeada. Com ela, é possível percorrer, filtrar, mapear e reduzir coleções de elementos de forma concisa e legível. A sintaxe de uma stream é a seguinte:

```
coleção.stream()
    .operação1()
    .operação2()
    .operação3()
    ...
```

Exemplificando

```
import java.util.ArrayList;
import java.util.List;

public class ExemploStream {
    public static void main(String[] args) {
        List<Integer> numeros = new ArrayList<>();
        numeros.add(1);
        numeros.add(2);
        numeros.add(3);
        numeros.add(4);
        numeros.add(5);

        List<Integer> pares = numeros.stream()
                                    .filter(n -> n % 2 == 0)
```

```
                                    .collect(Collectors.toList());
        System.out.println(pares);
    }
}
```

Existem inúmeras operações que podem ser realizadas em uma stream, como `filter()`, `map()`, `reduce()`, entre outras. A utilização de streams pode tornar o código mais legível e conciso, além de permitir a execução de operações em paralelo, o que pode melhorar a performance em alguns casos.