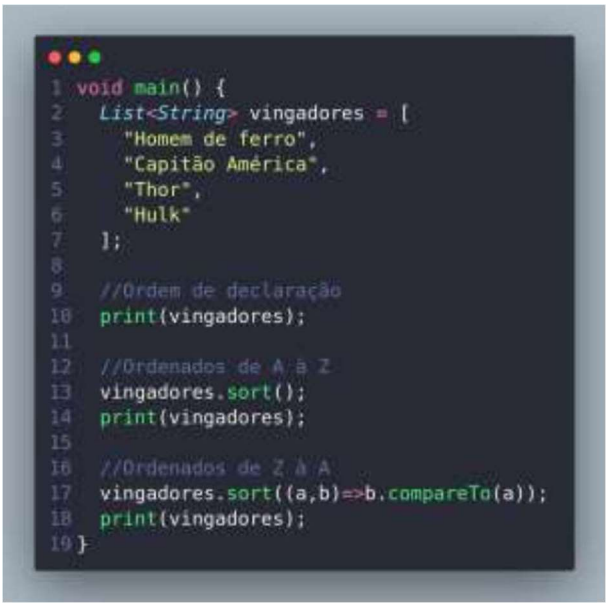
Método sort

O método sort, como o próprio nome já sugere, tem como objetivo ordenar os elementos da lista.

Caso o parâmetro de comparação seja omitido, a lista será ordenada de forma crescente (Caso seja uma lista numérica) e em ordem alfabética (Caso seja uma lista não numerada).

Também é possível informar um "modo de ordenação", fazendo uso do método compare da classe Comparable.

Parâmetro	Descrição
[modo de ordenação]	Método de comparação entre os elementos da lista



```
void main() {
  List<String> vingadores = [
    "Homem de ferro",
    "Capitão América",
    "Thor",
    "Hulk"
  ];

  //Ordem de declaração
  print(vingadores);

  //Ordenados de A à Z
  vingadores.sort();
  print(vingadores);

  //Ordenados de Z à A
  vingadores.sort((a,b)=>b.compareTo(a));
  print(vingadores);
}
```

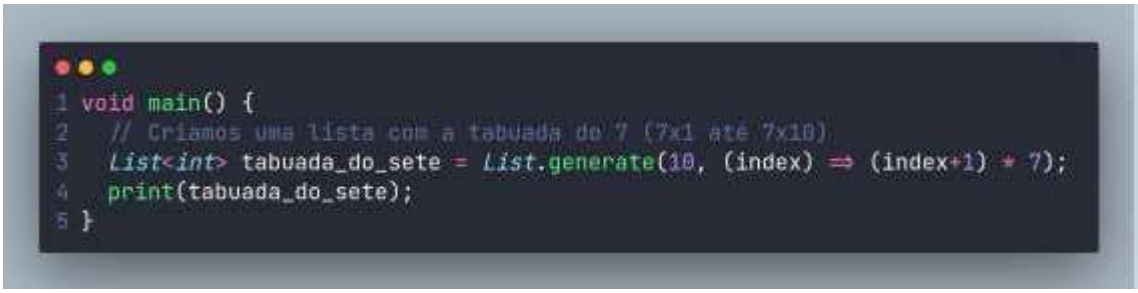
Método generate

O método generate, como o próprio nome já sugere, tem como objetivo gerar uma lista com um determinado número de elementos.

Com o uso deste método é possível criar uma lista preenchida com valores randômicos pré-configurados.

Muito útil quando desejamos testar uma implementação e necessitamos de dados fictícios para realizar os testes desejados.

Parâmetro	Descrição
Largura	Número de elementos a serem gerados
Gerador	Função que irá gerar os elementos
[Cultivável]	Valor booleano que define se a lista poderá ter mais elementos adicionados a ela. Se não informado o valor padrão é falso



```dart
void main() {
  // Criamos uma lista com a tabuada do 7 (7x1 até 7x10)
  List<int> tabuada_do_sete = List.generate(10, (index) => (index+1) * 7);
  print(tabuada_do_sete);
}
```

Método elementAt

O método elementAt retorna um elemento da lista com base em seu índice na mesma. Esse método é muito útil para casos em que desejamos recuperar algum elemento presente na lista para utilizá-lo para algum propósito.

Parâmetro	Descrição
Índice	Índice do elemento que desejamos recuperar

Com base no exemplo anterior, vamos adaptar nossa aplicação para receber um valor e mostrar o resultado deste na tabuada do 7.

```dart
void main() {
  print(valorNaTabuada(4));
}

String valorNaTabuada(int numero) {
  // Lembre-se as listas iniciam em 0
  int indice = numero - 1;
  // Criamos uma lista com a tabuada do 7 (7x1 até 7x10)
  List<int> tabuada_do_sete = List.generate(10, (index) => (index + 1) * 7);
  return "$numero x 7 = ${tabuada_do_sete.elementAt(indice)}";
}

```

Método forEach

O método forEach navega por cada elemento da lista, realizando uma operação a cada ciclo de interação. Podemos utilizar esse método para modificar cada elemento da lista de acordo com nosso desejo ou necessidade.

Parâmetro	Descrição
função	Função que desejamos executar a cada volta do ciclo. Essa NÃO deve retornar nenhum valor

```dart
void main() {
  List<int> numeros = [0, 2, 4, 6, 8, 10, 12];
  List<int> numerosAoQuadrado = List.empty(growable: true);

  numeros.forEach((numero) {
    numerosAoQuadrado.add(numero * numero);
  });

  print(numerosAoQuadrado);
}
```

Método toList

O método toList é utilizado para converter um objeto Iterable em um objeto List.

Para nosso exemplo, atualizaremos nosso exemplo anterior para que ele exiba os dados como um objeto List e não mais como um objeto Iterable.

```dart
void main() {
  List<int> numeros = List.generate(10, (index) => index + 1);

  // Imprimimos os números da lista
  print(numeros);

  // Filtramos os números pares
  Iterable<int> numerosPares = numeros.where((numero) => numero % 2 == 0);

  // Filtramos os números ímpares
  Iterable<int> numerosImpares = numeros.where((numero) => numero % 2 != 0);

  // Imprimimos números pares (Agora como lista)
  print("São números pares: ${numerosPares.toList()}");

  // Imprimimos números ímpare (Agora como lista)
  print("São números ímpares: ${numerosImpares.toList()}");
}

```