Recapitulando conceitos do Java

🔹 1. Leitura de dados rápida (Scanner, BufferedReader)

✅ Com `Scanner` (simples):

```
Scanner sc = new Scanner(System.in);
int x = sc.nextInt();
String nome = sc.next();
```

✅ Com `BufferedReader` (mais rápido):

```
BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
int x = Integer.parseInt(br.readLine());
String[] partes = br.readLine().split(" ");
```

🔹 2. Tipos primitivos

```
int, long, double, float, boolean, char
```

Use `long` para números grandes e `double` para casas decimais.

🔹 3. Vetores e Matrizes

Vetor:

```
int[] v = new int[5];
v[0] = 10;
```

Matriz:

```
int[][] mat = new int[3][3];
mat[0][1] = 7;
```

🔹 4. Laços de repetição

```
for (int i = 0; i < n; i++) {...}
while (condicao) {...}
```

🔹 5. Condicionais

```java
if (x > 0) {...}
else if (x == 0) {...}
else {...}
```

🔹 6. Funções (métodos)

```java
static int soma(int a, int b) {
    return a + b;
}
```

🔹 7. Strings e manipulação

```java
String s = "abc";
int tamanho = s.length();
char c = s.charAt(1);
String[] partes = s.split(" ");
```

🔹 8. Listas e Coleções (`ArrayList`, `HashSet`, `HashMap`)

```java
List<Integer> lista = new ArrayList<>();
lista.add(10);

Set<String> set = new HashSet<>();
Map<String, Integer> mapa = new HashMap<>();
```

🔹 9. Ordenação com `Collections.sort()`

```java
Collections.sort(lista); // crescente
lista.sort(Collections.reverseOrder()); // decrescente
```

Para ordenar objetos, use `Comparator`.

🔹 10. Tratamento de entrada múltipla

```java
String[] entrada = br.readLine().split(" ");
int a = Integer.parseInt(entrada[0]);
int b = Integer.parseInt(entrada[1]);
```

🔹 11. Algoritmos e estruturas de dados importantes

- Busca binária: `Collections.binarySearch(lista)`
- Fila: `Queue<Integer> fila = new LinkedList<>();`

- Pilha: `Stack<Integer> pilha = new Stack<>();`
- Ordenação personalizada: `Collections.sort(lista, Comparator.comparing(...))`

🔹 12. Classes e Objetos (caso precise ordenar por atributos)

```
class Pessoa {
    String nome;
    int idade;
    Pessoa(String nome, int idade) {
        this.nome = nome;
        this.idade = idade;
    }
}
```

Vetores e matrizes no Java

🔹 1. Arrays (vetores)

✅ Declaração e inicialização:

```
int[] numeros = new int[5];         // vetor com 5 posições (valores padrão = 0)
int[] valores = {1, 2, 3, 4, 5};    // inicialização direta
```

✅ Atribuição e acesso:

```
numeros[0] = 10;
System.out.println(numeros[0]);  // 10
```

✅ Percorrer com `for`:

```
for (int i = 0; i < numeros.length; i++) {
    System.out.println(numeros[i]);
}
```

🔹 2. Matrizes (arrays 2D)

✅ Declaração e inicialização:

```java
int[][] matriz = new int[3][4];  // 3 linhas, 4 colunas (todos zero)
```

✅ **Atribuição:**

```java
matriz[0][1] = 7;
System.out.println(matriz[0][1]);  // 7
```

✅ **Percorrer com `for` aninhado:**

```java
for (int i = 0; i < matriz.length; i++) {
    for (int j = 0; j < matriz[i].length; j++) {
        System.out.print(matriz[i][j] + " ");
    }
    System.out.println();
}
```

🔹 3. Matriz com valores fixos:

```java
int[][] fixos = {
    {1, 2, 3},
    {4, 5, 6}
};
```

🔹 4. Usar `ArrayList` (lista dinâmica)

✅ **Declaração:**

```java
List<Integer> lista = new ArrayList<>();
```

✅ **Operações básicas:**

```java
lista.add(10);
lista.add(20);
System.out.println(lista.get(0));  // 10
System.out.println(lista.size()); // 2
```

✅ **Percorrer com `for-each`:**

```java
for (int valor : lista) {
    System.out.println(valor);
}
```

🔹 5. Lista de listas (`ArrayList<ArrayList<Integer>>`)

✅ Exemplo de matriz dinâmica:

```
List<List<Integer>> matriz = new ArrayList<>();

for (int i = 0; i < 3; i++) {
    matriz.add(new ArrayList<>());
    for (int j = 0; j < 4; j++) {
        matriz.get(i).add(i * j);
    }
}
```

✅ Acessar elemento específico:

```
System.out.println(matriz.get(1).get(2));  // valor da linha 1, coluna 2
```

Ordenando Listas

🔹 1. Ordenar vetor (array) com `Arrays.sort()`

✅ Importar:

```
import java.util.Arrays;
```

✅ Exemplo com vetor de inteiros:

```
int[] numeros = {5, 3, 8, 1, 2};
Arrays.sort(numeros);  // ordena em ordem crescente

System.out.println(Arrays.toString(numeros)); // [1, 2, 3, 5, 8]
```

🔹 2. Ordenar em ordem decrescente (com `Integer[]`)

Para usar ordem decrescente, precisa usar `Integer` (classe) ao invés de `int` (primitivo):

```
Integer[] numeros = {5, 3, 8, 1, 2};
Arrays.sort(numeros, Collections.reverseOrder());

System.out.println(Arrays.toString(numeros)); // [8, 5, 3, 2, 1]
```

🔹 3. Ordenar `ArrayList` com `Collections.sort()`

✅ Importar:

```
import java.util.ArrayList;
import java.util.Collections;
```

✅ Exemplo:

```
List<Integer> lista = new ArrayList<>();
lista.add(4);
lista.add(1);
lista.add(7);

Collections.sort(lista); // crescente
System.out.println(lista); // [1, 4, 7]

Collections.sort(lista, Collections.reverseOrder()); // decrescente
System.out.println(lista); // [7, 4, 1]
```

🔹 4. Ordenar matriz por linha ou coluna (customizado)

✅ Ordenar cada linha da matriz:

```
int[][] matriz = {
    {3, 1, 2},
    {9, 7, 8}
};

for (int i = 0; i < matriz.length; i++) {
    Arrays.sort(matriz[i]);
}
```

🔹 5. Ordenar objetos com `Comparator`

Se você tiver uma lista de objetos (como `Pessoa`), pode ordenar por atributos:

```
class Pessoa {
    String nome;
    int idade;
    Pessoa(String nome, int idade) {
        this.nome = nome;
        this.idade = idade;
    }
```

```
}

// Em algum lugar do código:
List<Pessoa> pessoas = new ArrayList<>();
pessoas.add(new Pessoa("Ana", 20));
pessoas.add(new Pessoa("Beto", 18));

pessoas.sort(Comparator.comparing(p -> p.idade)); // crescente por idade
```

Formatar valores de saida

🔹 1. Usando `System.out.printf()`

✅ Sintaxe básica:

```
System.out.printf("Texto %formato", valor);
```

🧮 Exemplos práticos

✅ 1. Duas casas decimais (%.2f):

```
double x = 3.14159;
System.out.printf("%.2f\n", x);  // Saída: 3.14
```

✅ 2. Número com largura fixa:

```
int x = 42;
System.out.printf("%5d\n", x);   // Saída: '   42' (com espaços à esquerda)
```

✅ 3. Alinhar texto (esquerda/direita):

```
String nome = "Ana";
System.out.printf("%-10s OK\n", nome);  // 'Ana       OK' (alinhado à esquerda)
```

✅ 4. Exibir múltiplos valores formatados:

```
String nome = "Bob";
int idade = 30;
double nota = 8.75;

System.out.printf("Nome: %s | Idade: %d | Nota: %.1f\n", nome, idade, nota);
```

```
// Saída: Nome: Bob | Idade: 30 | Nota: 8.8
```

🔹 2. Usar `String.format()` para armazenar o texto formatado

```
String linha = String.format("Valor: %.2f", 5.6789);
System.out.println(linha);  // Valor: 5.68
```

🔹 3. Formatar valores monetários ou com vírgula

Você pode usar `Locale` e `NumberFormat`:

```
import java.text.NumberFormat;
import java.util.Locale;

double preco = 12345.67;
NumberFormat nf = NumberFormat.getCurrencyInstance(new Locale("pt",
"BR"));
System.out.println(nf.format(preco)); // R$ 12.345,67
```

Leitura de múltiplos valores no Java

🔹 Como ler múltiplos valores em uma linha com `Scanner`

✅ Importar o Scanner:

```
import java.util.Scanner;
```

✅ Exemplo: ler 3 inteiros em uma única linha

```
Scanner sc = new Scanner(System.in);

int a = sc.nextInt();
int b = sc.nextInt();
int c = sc.nextInt();
```

Se a entrada for:

```
10 20 30
```

A leitura funciona como o `input().split()` do Python.

🔹 Ler N valores de uma linha (usando `for` ou array)

```
int[] numeros = new int[5];
for (int i = 0; i < 5; i++) {
    numeros[i] = sc.nextInt();
}
```

🔹 Ler uma linha inteira e dividir com `.split(" ")`

```
String linha = sc.nextLine();           // Lê a linha completa
String[] partes = linha.split(" ");     // Divide nos espaços

int a = Integer.parseInt(partes[0]);
int b = Integer.parseInt(partes[1]);
```

🔹 Ler múltiplas linhas (ex: matriz 3x3)

```
int[][] matriz = new int[3][3];
for (int i = 0; i < 3; i++) {
    String[] valores = sc.nextLine().split(" ");
    for (int j = 0; j < 3; j++) {
        matriz[i][j] = Integer.parseInt(valores[j]);
    }
}
```

🔹 Ler diferentes tipos de dados em sequência

```
String nome = sc.next();
int idade = sc.nextInt();
double altura = sc.nextDouble();
```

Entrada:

```
Ana 22 1.65
```

Dica importante:

Se você misturar `sc.nextLine()` com `sc.nextInt()` ou `sc.next()`, pode precisar de um `sc.nextLine()` extra para **consumir a quebra de linha**.

<u>Usos do while e Switch</u>

🔁 `while` — Repetição com condição

✅ Sintaxe:

```
while (condição) {
    // bloco de código
}
```

✅ Exemplo: contar de 1 a 5

```
int i = 1;
while (i <= 5) {
    System.out.println(i);
    i++;
}
```

⚠️ Cuidado com **loops infinitos**: sempre garanta que a condição **muda dentro do loop**.

🌀 Uso comum: ler entradas até uma condição

```
Scanner sc = new Scanner(System.in);
int x = sc.nextInt();

while (x != 0) {
    System.out.println("Você digitou: " + x);
    x = sc.nextInt(); // atualiza para sair do loop se for 0
}
```

🔀 `switch` — Escolha entre opções

✅ Sintaxe:

```
switch (variável) {
    case valor1:
        // ação
        break;
    case valor2:
        // ação
        break;
    default:
        // ação padrão
}
```

✅ Exemplo: dias da semana

```
int dia = 3;
```

```java
switch (dia) {
    case 1:
        System.out.println("Domingo");
        break;
    case 2:
        System.out.println("Segunda");
        break;
    case 3:
        System.out.println("Terça");
        break;
    default:
        System.out.println("Outro dia");
}
```

🧠 `break` impede que os próximos `case` sejam executados em "efeito cascata".

🧪 Pode ser usado com `String` também:

```java
String comando = "salvar";

switch (comando) {
    case "abrir":
        System.out.println("Abrindo arquivo...");
        break;
    case "salvar":
        System.out.println("Salvando...");
        break;
    default:
        System.out.println("Comando inválido");
}
```

📌 Quando usar:

Recurso	Quando usar
`while`	Repetições com condição (ex: até entrada ser "0")
`switch`	Múltiplas decisões baseadas em 1 variável