**ESTUDOS - JAVA**

**Memória Buffer**

**Memória Buffer** é um espaço temporário onde dados são armazenados enquanto estão sendo movidos de um lugar para outro. No caso do nosso programa, o buffer é usado para armazenar temporariamente o que você digita no teclado antes de o programa ler esses dados.

Quando você digita algo e pressiona "ENTER", a informação vai para o buffer. O programa, então, lê esses dados do buffer. Só que, às vezes, o buffer pode "guardar" coisas que o programa não esperava, como o "ENTER" que sobra quando usamos nextDouble(). Por isso, a gente precisa "limpar" o buffer em alguns casos, como adicionando um nextLine().

**Classes Wrappers**

**Classes Wrappers** são classes em Java que "embrulham" tipos primitivos, como int, double, char, etc., para que você possa usá-los como objetos. Em Java, os tipos primitivos não são objetos, então, se você precisar trabalhar com eles em situações que exigem um objeto (como em coleções ArrayList), usa-se uma classe wrapper.

Alguns exemplos de classes wrappers:

* Para o tipo primitivo int, temos a classe Integer.
* Para double, temos a classe Double.
* Para char, temos a classe Character.

Essas classes ajudam a fazer conversões e manipulações extras que os tipos primitivos não conseguem fazer sozinhos.

**Resumo**

* **Buffer**: um espaço temporário onde dados são armazenados antes de serem usados pelo programa.
* **Wrapper**: uma classe que "transforma" tipos básicos (como int e double) em objetos para oferecer mais funcionalidades.

o nextLine limpa a memoria, tirando o enter?

Sim, exatamente! 😊

Quando você usa o nextLine() logo após um nextDouble(), ele "limpa" o **ENTER** que ficou sobrando no buffer. O nextLine() lê toda a linha que está no buffer até o próximo "ENTER", então, nesse caso, ele captura apenas o "ENTER" que estava sobrando e descarta, deixando o buffer vazio para a próxima entrada.

Assim, quando você pede o próximo valor (como o nome do aluno), o programa espera realmente pela nova entrada, sem interferências do "ENTER" anterior.

**Código com o Erro (Sem o nextLine() para limpar o buffer)**

import java.util.Scanner;

public class TesteErroBuffer {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Entre com a nota1: ");

double nota1 = sc.nextDouble();

System.out.print("Entre com a nota2: ");

double nota2 = sc.nextDouble();

// Aqui, o programa vai ler o ENTER deixado pelo último nextDouble() em vez de esperar pelo nome.

System.out.print("Entre com um nome: ");

String nome = sc.nextLine();

double media = (nota1 + nota2) / 2;

System.out.println("A média do aluno " + nome + " é " + media);

sc.close();

}

}

**Veja como ficaria o código com essa correção:**

System.out.print("Entre com a nota1: ");

double nota1 = sc.nextDouble();

sc.nextLine(); // limpa o ENTER que ficou no buffer

System.out.print("Entre com a nota2: ");

double nota2 = sc.nextDouble();

sc.nextLine(); // limpa o ENTER que ficou no buffer

System.out.print("Entre com um nome: ");

String nome = sc.nextLine();

**System.out.print("Entre com a nota1: "); double nota1 = sc.nextDouble(); sc.nextLine(); System.out.print("Entre com a nota1: "); double nota1 = Double.parseDouble(sc.nextLine()); AMBOS FAZEM A MESMA COISA?**

**Os dois trechos de código fazem a mesma coisa no sentido de obter um número decimal (do tipo double) como entrada do usuário, mas eles funcionam de maneiras um pouco diferentes. Vamos ver as diferenças:**

**Código 1: Usando nextDouble() com nextLine() para limpar o buffer**

java

Copiar código

System.out.print("Entre com a nota1: ");

double nota1 = sc.nextDouble();

sc.nextLine(); // Limpa o ENTER que ficou no buffer

**Explicação**:

* nextDouble() lê apenas o número digitado e deixa o "ENTER" no buffer.
* O sc.nextLine() adicional é necessário para limpar o "ENTER" do buffer, evitando problemas com leituras subsequentes de texto.

**Código 2: Usando nextLine() com Double.parseDouble()**

java

Copiar código

System.out.print("Entre com a nota1: ");

double nota1 = Double.parseDouble(sc.nextLine());

**Explicação**:

* sc.nextLine() lê a linha completa, incluindo o número e o "ENTER".
* Double.parseDouble(...) converte a String lida para um double.
* Como nextLine() já consome o "ENTER", não é necessário limpar o buffer com uma linha adicional.

**Diferenças**

* **Buffer**: O segundo método (nextLine() com Double.parseDouble()) evita o problema do "ENTER" no buffer, então você não precisa de um nextLine() extra.
* **Simplicidade**: O segundo método é mais simples e direto, já que nextLine() consome toda a linha de uma vez.

**Conclusão**

Ambos os métodos resultam na mesma variável nota1 com um valor double, mas o segundo método (nextLine() com Double.parseDouble()) é mais seguro e prático para evitar problemas de buffer.