using the `Scanner` class for user input in Java

Detailed Overview of User Inputs with Scanner

1. Importing Scanner:

To use the `Scanner` class, you must import it from the `java. Util package: Import java. Util.Scanner;

1. Creating a Scanner Object:

You create a `Scanner` object to read input. The most common way is to read from the console:

Scanner scanner = new Scanner(System.in);

1. Types of Input Methods:

The `Scanner` class provides several methods to read different types of input:

String Input:

next(): Reads the next token (word) from the input.

String word = scanner.next();

NextLine (): Reads the entire line of input, including spaces.

String line = scanner.nextLine(); Numeric Input: nextInt(): Reads an integer value. Throws `InputMismatchException` if the input is not an integer. int number = scanner.nextInt();

nextDouble(): Reads a double value. Throws `InputMismatchException` if the input is not a double. double decimal = scanner.nextDouble();

nextFloat(): Reads a float value.

float floatValue = scanner.nextFloat(); nextLong(): Reads a long value.

long longValue = scanner.nextLong(); Boolean Input:

nextBoolean() : Reads a boolean value (`true` or `false`).

boolean flag = scanner.nextBoolean();

1. Common Scenarios:

Reading Multiple Inputs: You can read multiple inputs sequentially. Be careful when mixing `nextInt()` or `nextDouble()` with `nextLine()` because `nextInt()` and `nextDouble()` do not consume the newline character after pressing Enter.

Flushing the Input Buffer: After reading a number, if you want to read a string (using `nextLine()`), you may need to consume the leftover newline character: scanner.nextLine(); // Consume the newline

String name = scanner.nextLine();

1. Handling Input Mismatches:

It’s essential to handle exceptions when reading user inputs. Use a `try-catch` block to catch potential exceptions:

try { int number = scanner.nextInt();

} catch (InputMismatchException e) {

System.out.println("Invalid input! Please enter an integer."); scanner.next(); // Clear the invalid input

}

1. Closing the Scanner:

Always close the `Scanner` when you're done to free up resources:

scanner.close();

Example Program

Here’s an example program that demonstrates various types of user inputs using `Scanner`: import java. util.Scanner; import java. util.InputMismatchException;

public class UserInputExample {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Reading a string

System.out.print("Enter your name: ");

String name = scanner.nextLine();

// Reading an integer

int age = 0; while (true) {

try {

System.out.print("Enter your age: "); age = scanner.next(); break; // Exit the loop if input is valid

} catch (InputMismatchException e) {

System.out.println("Invalid input! Please enter a valid age."); scanner.next(); // Clear the invalid input

}

}

// Reading a double

System.out.print("Enter your height in meters: "); double height = scanner.nextDouble();

// Reading a boolean

System.out.print("Are you a student? (true/false): "); boolean isStudent = scanner.nextBoolean();

// Displaying the input

System.out.println("Hello, " + name + "! You are " + age + " years old, " +

"your height is " + height + " meters, and it is " + isStudent + " that you are a student.");

// Closing the scanner scanner.close();

}

}

Key Takeaways

* The `Scanner` class is versatile for reading various types of user input.
* Be mindful of input types and handle exceptions to prevent runtime errors.
* Always close the scanner to avoid resource leaks.