Java Input and Output

This guide explains how to perform **Input and Output (I/O)** operations in Java. It covers formatted output, reading various types of input (string, integer, multiline, and character), with code examples and syntax for each operation. Additionally, we will discuss the **import statement** needed to use the `Scanner` class for input.

---

1. Input and Output in Java

Java provides multiple ways to handle input and output operations. The primary classes used for I/O in Java are:

- `System.out`: For output (printing data to the console).
- `System.in`: For input (reading data from the console).

We use the `Scanner` class (from the `java.util` package) to read input from the user. It can read different types of input, such as integers, strings, characters, and more.

Import Statement for `Scanner`:

To use the `Scanner` class, you need to import it from the `java.util` package:

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

---

2. Output in Java (Formatted Output)

Using `System.out.print()`, `System.out.println()`, and `System.out.printf()`

- `System.out.print()`: Prints the output without a newline.
- `System.out.println()`: Prints the output with a newline.
- `System.out.printf()`: Prints output with specific formatting (like C-style printf).

Syntax for Formatted Output (`printf`):

```
System.out.printf("format specifier", variable);
```

- **format specifier**: A placeholder for data (e.g., `%d` for integers, `%s` for strings, `%f` for floating-point numbers).

Example of Formatted Output:

```
import java.util.Scanner;

class FormattedOutput {
    public static void main(String[] args) {
        int age = 25;
        double price = 10.50;
        String name = "Alice";

        // Print using printf with formatting
        System.out.printf("Name: %s\n", name);
        System.out.printf("Age: %d years\n", age);
        System.out.printf("Price: %.2f dollars\n", price);
    }
}
```

Output:

```
Name: Alice
Age: 25 years
Price: 10.50 dollars
```

---

3. Input in Java (Using `Scanner` Class)

To read input in Java, we use the `Scanner` class. Here's how you can read different types of input.

3.1. Reading String Input

You can read a string (including spaces) from the user using `nextLine()`.

Syntax for String Input:

```
String input = scanner.nextLine();
```

Example of Reading String Input:

```java
import java.util.Scanner;

class StringInput {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter your name:");

        // Read a full line of input
        String name = scanner.nextLine();
        System.out.println("Hello, " + name + "!");
    }
}
```

Output:

```
Enter your name:
Alice
Hello, Alice!
```

3.2. Reading Integer Input

To read an integer, use the `nextInt()` method.

Syntax for Integer Input:

```java
int number = scanner.nextInt();
```

Example of Reading Integer Input:

```java
import java.util.Scanner;

class IntegerInput {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter your age:");

        // Read an integer
        int age = scanner.nextInt();
        System.out.println("You are " + age + " years old.");
    }
}
```

Output:

```
Enter your age:
25
You are 25 years old.
```

3.3. Reading Character Input

To read a single character, you can use `next().charAt(0)`.

Syntax for Character Input:

```
char ch = scanner.next().charAt(0);
```

Example of Reading Character Input:

```
import java.util.Scanner;

class CharacterInput {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter a character:");

        // Read a character
        char character = scanner.next().charAt(0);
        System.out.println("You entered: " + character);
    }
}
```

Output:

```
Enter a character:
A
You entered: A
```

3.4. Reading Multiline Input

To read multiple lines, use the `nextLine()` method in a loop. It will read input until the user presses **Enter**.

Example of Multiline Input:

```java
import java.util.Scanner;

class MultilineInput {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter multiple lines of text (enter 'STOP' to end):");

        // Read multiple lines
        String line;
        while (!(line = scanner.nextLine()).equals("STOP")) {
            System.out.println("You entered: " + line);
        }
    }
}
```

Output:

```
Enter multiple lines of text (enter 'STOP' to end):
Hello, how are you?
You entered: Hello, how are you?
STOP
```

---

4. Input Validation (Optional)

Sometimes, it's important to validate user input. Here's an example of how you can validate that the user enters a valid integer.

Example of Input Validation:

```java
import java.util.Scanner;

class InputValidation {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int number;

        // Input validation loop
        while (true) {
            System.out.println("Enter a valid integer:");

            if (scanner.hasNextInt()) {
                number = scanner.nextInt();
                break;  // Exit loop if input is valid
            } else {
                System.out.println("That's not a valid integer! Please try again.");
```

```
                scanner.next();  // Clear the invalid input
            }
        }

        System.out.println("You entered the number: " + number);
    }
}
```

Output:

```
Enter a valid integer:
Hello
That's not a valid integer! Please try again.
Enter a valid integer:
25
You entered the number: 25
```

---

5. Summary

- **Formatted Output**: Use `System.out.printf()` for formatted printing with placeholders for values like `%d`, `%s`, and `%f`.
- **Reading String Input**: Use `scanner.nextLine()` for reading strings (including spaces).
- **Reading Integer Input**: Use `scanner.nextInt()` for reading integers.
- **Reading Character Input**: Use `scanner.next().charAt(0)` to read a single character.
- **Multiline Input**: Use `scanner.nextLine()` in a loop to read multiple lines until a stop condition is met.
- **Input Validation**: You can validate input using methods like `scanner.hasNextInt()` to ensure correct data types.
