JAVA 21 OUTPUT STATEMENTS REFERENCE

1. THE THREE PRINT METHODS:

System.out.println()

Prints its argument followed by a newline.
Each call starts output on a fresh line.
Called with no arguments, it prints a blank line.

```
System.out.println("Hello");   // Hello
System.out.println();          // blank line
```

System.out.print()

Prints its argument WITHOUT a newline.
Subsequent output continues on the same line.

```
System.out.print("Hello ");
System.out.print("World");     // Hello World  (on one line)
```

System.out.printf()

Prints formatted output using format specifiers.
Use %n for newline inside printf (preferred over \n).

```
System.out.printf("Hello, %s! You are %d years old.%n",
"Alice", 30);
```

2. ESCAPE SEQUENCES:

Escape sequences are special characters inside strings.
They always begin with a backslash \.

`\n`	newline
`\t`	tab (useful for basic column spacing)
`\\`	a literal backslash character
`\"`	a literal double-quote character

Examples:

```
System.out.println("Line one\nLine two");
System.out.println("Name\tAge\tCity");
System.out.println("C:\\Users\\Alice");
System.out.println("She said, \"Hello!\"");
```

3. STRINGS:

A String holds a sequence of characters enclosed in double quotes.
String is an object type, not a primitive.

Declaration:
```
String name = "Alice";
```

Concatenation with +:
```
String full = "Alice" + " " + "Smith";  // Alice Smith
```

When a non-String value is concatenated with +, Java automatically calls .toString() on it to convert it to a String first.

Useful String methods:
```
name.toUpperCase()    // ALICE
name.toLowerCase()    // alice
name.length()         // 5
name.charAt(0)        // A
```

Text Blocks (Java 15+):
Use triple double-quotes for multi-line strings.
Leading whitespace is stripped based on indentation.

```
String profile = """
        Name: Alice
        Role: Developer
        """;
```

4. NUMERIC DATA TYPES:

INTEGER TYPES (whole numbers)

byte	8-bit	range: -128 to 127
short	16-bit	range: -32,768 to 32,767
int	32-bit	range: ~-2.1 billion to 2.1 billion
long	64-bit	very large whole numbers

long requires an L suffix on the literal:
```
long population = 8_000_000_000L;
```

Underscores can be used in numeric literals for readability:
```
int million = 1_000_000;
```

FLOATING-POINT TYPES (decimal numbers)

float   32-bit   less precise
double 64-bit   more precise (default choice for decimals)

```
double price = 19.99;
```

float requires an f suffix on the literal:

```
float price = 9.99f;
```

When in doubt, use double.

BOOLEAN TYPE

boolean holds only true or false.
Commonly printed during debugging.

```
 boolean isStudent = true;
System.out.println(isStudent);   // true
```

5. COMMON BEGINNER GOTCHAS:

GOTCHA 1: Integer Division
When both operands are integer types, Java discards the decimal.

```
 int a = 5, b = 2;
System.out.println(a / b);          // prints 2, not 2.5!
```

Fix: cast one operand to double first.

```
System.out.println((double) a / b); // prints 2.5
```

GOTCHA 2: The String Concatenation Trap
The + operator is evaluated left to right.
Once Java sees a String, all following + operators become concatenation (joining), not addition.

```
 int x = 3, y = 4;
System.out.println("Value: " + x + y);
            // prints "Value: 34" !!
```

Fix: use parentheses to force addition first.

```
System.out.println("Value: " + (x + y));  // prints "Value: 7"
```

6. PRINTF FORMAT SPECIFIERS:

Format specifiers are placeholders inside a printf format string.
Each starts with % and ends with a type letter.

%s	String
%d	integer (byte, short, int, long)
%f	floating-point (float or double)
%b	boolean
%n	newline (preferred inside printf)

Precision for floating-point:
%.2f means: print with exactly 2 decimal places

Multiple specifiers in one call:
```
System.out.printf("Name: %s, Age: %d, GPA: %.2f%n", name, age, gpa);
```
Values are matched to specifiers left to right.

Comma separator for large numbers:
%,.2f adds thousands comma separator, e.g. 18,500.50

7. WIDTH SPECIFIERS:

A width specifier reserves a fixed number of characters for a value.
Put a number between % and the type letter.

Format: % [flags] [width] [.precision] type

%13d	integer, right-aligned in a 13-character field
%-13d	integer, LEFT-aligned in a 13-character field (minus = left)
%10.2f	float, right-aligned in a 10-char field, 2 decimal places
%-15s	string, left-aligned in a 15-character field

By default, values are RIGHT-aligned, padded with spaces on the left.
Add a minus sign (-) flag to switch to left-alignment.

If the value is wider than the field, Java prints it in full anyway.
Data is NEVER truncated to fit the width.

Examples:
```
System.out.printf("%13d%n", 240);      //           240
System.out.printf("%-13d|%n", 240);    // 240          |
System.out.printf("%10.2f%n", 3.14);   //       3.14
System.out.printf("%-10s|%n", "hi");   // hi        |
```

8. ALIGNED COLUMNS WITH WIDTH SPECIFIERS:

Use the same format string for every row to create aligned tables.

```
      System.out.printf("%-12s %6s %8s%n", "Student", "Score", "Average");
      System.out.println("-".repeat(28));
      System.out.printf("%-12s %6d %8.2f%n", "Alice",     95, 91.75);
      System.out.printf("%-12s %6d %8.2f%n", "Bob",      100, 98.50);
      System.out.printf("%-12s %6d %8.2f%n", "Charlotte",  88, 85.33);
```

Output:

```
  Student            Score            Average
  ----------------------------------------------------------------
  Alice              95               91.75
  Bob                100              98.50
  Charlotte          88               85.33
```

9. STRING.FORMATTED()

String.formatted() builds a formatted String without printing it.
Useful when you want to store the result in a variable first.

```
      String label = "Score: %d out of 100".formatted(95);
      System.out.println(label);   // Score: 95 out of 100
```

Pairs cleanly with Text Blocks:

```
  String report = """
            Name: %s
            Age:  %d
            GPA:  %.2f
            """.formatted(name, age, gpa);
    System.out.println(report);
```

String.format() is an older equivalent that works the same way:
  String label = String.format("Score: %d", 95);

10. WHEN TO USE EACH APPROACH:

println + concatenation
Good for: quick output, simple debugging, single values.

```
System.out.println("Name: " + name);
```

printf / String.formatted()
Good for: precise formatting, decimal places, aligned columns, mixing multiple values of different types cleanly.

```
System.out.printf("GPA: %.2f%n", gpa);
```

Text Blocks
Good for: multi-line output, structured reports, embedded content.
Available in Java 15 and later.

END OF REFERENCE