Module 03 — Java I/O

ITP 120 — Java Programming I

Northern Virginia Community College | Randy Michak

🎯 Learning Objectives

- Use Scanner methods to read different data types from the keyboard
- Explain the nextLine() buffer problem and apply the correct fix
- Display dialogs and collect input using JOptionPane
- Format output with `printf()` and `String.format()`
- Convert between numeric types using widening and narrowing casts
- Call common Math class methods for calculations and random numbers

1. Scanner in Depth

You already know how to create a Scanner and read a single integer. This section covers every reading method you will use regularly.

```
import java.util.Scanner;

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

Scanner Reading Methods

Method	Reads	Example
`nextInt()`	int value	`int age = input.nextInt();`
`nextDouble()`	double value	`double gpa = input.nextDouble();`
`nextLong()`	long value	`long pop = input.nextLong();`

Method	Reads	Example
`nextBoolean()`	true or false	`boolean ok = input.nextBoolean();`
`next()`	One word (stops at whitespace)	`String w = input.next();`
`nextLine()`	Entire line including spaces	`String line = input.nextLine();`

```
import java.util.Scanner;

public class ScannerDemo {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);

        System.out.print("Enter your age: ");
        int age = input.nextInt();

        System.out.print("Enter your GPA: ");
        double gpa = input.nextDouble();

        System.out.print("Enter your first name: ");
        String firstName = input.next();  // one word only

        System.out.println("Age: " + age);
        System.out.println("GPA: " + gpa);
        System.out.println("Name: " + firstName);
    }
}
```

```
Enter your age: 20
Enter your GPA: 3.75
Enter your first name: Alex
Age: 20
GPA: 3.75
Name: Alex
```

> **💡 Tip: next() vs nextLine()**
>
> `next()` stops at the first whitespace. If the user types "Randy Michak", `next()` only captures "Randy". Use `nextLine()` when you need the full line including spaces.

> **🧠 Think of It This Way**
>
> Scanner is like a grocery store conveyor belt. Each call to `nextInt()` or `next()` picks up the next item and advances the belt forward. It never goes backward.

2. The nextLine() Problem

This is the most common Scanner bug beginners write. It catches almost everyone the first time.

What Goes Wrong

When a user types a number and presses Enter, the buffer holds two things: the number, and the newline character (`\n`) from the Enter key. `nextInt()` reads the number and stops — it leaves that `\n` sitting in the buffer. The next `nextLine()` immediately reads that leftover newline and returns an empty string.

```
import java.util.Scanner;

public class NextLineBug {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);

        System.out.print("Enter your age: ");
        int age = input.nextInt();       // reads 20, leaves '\n' in buffer

        System.out.print("Enter your name: ");
        String name = input.nextLine();  // reads '\n' -- user never types!

        System.out.println("Age: " + age);
        System.out.println("Name: [" + name + "]");  // prints empty brackets
    }
}
```

```
Enter your age: 20
Enter your name:
Age: 20
Name: []
```

> ⚠️ **Common Mistake**
>
> Any time you call `nextInt()`, `nextDouble()`, `nextLong()`, or `nextBoolean()` and then call `nextLine()` right after it, you will hit this bug. Every single time.

The Fix

Add one extra `nextLine()` call immediately after the numeric read. Its only job is to consume — throw away — that leftover newline.

```java
import java.util.Scanner;

public class NextLineFix {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);

        System.out.print("Enter your age: ");
        int age = input.nextInt();
        input.nextLine();   // THE FIX: consume the leftover newline

        System.out.print("Enter your name: ");
        String name = input.nextLine(); // now this works correctly

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

```
Enter your age: 20
Enter your name: Randy Michak
Age: 20
Name: Randy Michak
```

💡 Rule of Thumb

Whenever you use `nextInt()` or `nextDouble()` and you might read a full line afterward, add `input.nextLine();` immediately below it. It costs nothing and prevents the bug every time.

🔧 Try It

Type the buggy version first and confirm the name comes out blank. Then add the one-line fix and run again. Seeing the bug yourself is the best way to remember the fix.

3. Dialog Boxes: JOptionPane

Not every Java program runs in a terminal. `JOptionPane` gives you pop-up dialog boxes for desktop use. It is part of `javax.swing`, which is bundled with Java — no extra libraries needed.

```
import javax.swing.JOptionPane;
```

Showing a Message

`showMessageDialog()` pops up a window with text and an OK button. Pass `null` as the first argument to center it on the screen.

```
JOptionPane.showMessageDialog(null, "Hello, World!");
```

Getting Input

`showInputDialog()` shows a text field and returns whatever the user typed as a `String`.

```
String name = JOptionPane.showInputDialog("What is your name?");
JOptionPane.showMessageDialog(null, "Hello, " + name + "!");
```

Converting Input to a Number

Because `showInputDialog()` always returns a String, you need to convert it when you expect a number.

```
import javax.swing.JOptionPane;

public class DialogDemo {
    public static void main(String[] args) {
        String ageStr = JOptionPane.showInputDialog("Enter your age:");
        int age = Integer.parseInt(ageStr);     // String to int

        String gpaStr = JOptionPane.showInputDialog("Enter your GPA:");
        double gpa = Double.parseDouble(gpaStr); // String to double

        JOptionPane.showMessageDialog(null,
            "Age: " + age + "\nGPA: " + gpa);
    }
}
```

📖 **Key Terms**

- **Integer.parseInt(str)** — converts a String like "42" into the int 42
- **Double.parseDouble(str)** — converts a String like "3.14" into the double 3.14

⚠️ **Watch Out**

If the user types "abc" or leaves the field blank, `parseInt()` and `parseDouble()` will crash with a `NumberFormatException`. Exception handling is covered later in the course.

4. Formatted Output: printf()

`System.out.printf()` gives you precise control over how output looks. You define a format pattern and plug in values.

```
System.out.printf(formatString, value1, value2, ...);
```

Format Specifiers

Specifier	Meaning	Example	Output
`%d`	Integer	`printf("%d", 42)`	42
`%f`	Decimal (float/double)	`printf("%f", 3.14)`	3.140000
`%s`	String	`printf("%s", "Java")`	Java
`%n`	Newline (platform-safe)	Use instead of \n inside printf	
`%.2f`	Decimal, 2 places	`printf("%.2f", 3.876)`	3.88
`%10.2f`	10 wide, 2 decimal places, right-aligned	`printf("%10.2f", 3.5)`	right-justified in 10 chars
`%-10s`	String, 10 wide, left-aligned	`printf("%-10s", "Hi")`	left-justified in 10 chars

```
public class PrintfDemo {
    public static void main(String[] args) {
        String name = "Alice";
        int age = 21;
        double gpa = 3.876;

        System.out.printf("Name: %s%n", name);
        System.out.printf("Age:  %d%n", age);
        System.out.printf("GPA:  %.2f%n", gpa);  // rounds to 2 decimal places
    }
}
```

```
Name: Alice
Age:  21
GPA:  3.88
```

Column Alignment

```
System.out.printf("%-15s %5d %8.2f%n", "Alice",   21, 3.876);
System.out.printf("%-15s %5d %8.2f%n", "Bob",     19, 2.950);
System.out.printf("%-15s %5d %8.2f%n", "Charlie", 22, 3.100);
```

```
Alice              21     3.88
Bob                19     2.95
Charlie            22     3.10
```

> 🧠 **Think of It This Way**
>
> The format string is a form with blanks to fill in. Each `%` placeholder is a blank, and the values after the comma fill them in left to right.

> 💡 **Use %n in printf**
>
> Inside `printf()` format strings, use `%n` for newlines. It outputs the correct newline character for whatever operating system is running the program.

5. String.format()

`String.format()` uses the exact same format specifiers as `printf()`, but instead of printing it *returns* a formatted String you can store or use anywhere.

```
String message = String.format("Name: %-10s  GPA: %.2f", "Alice", 3.876);
System.out.println(message);
```

```
Name: Alice       GPA: 3.88
```

This is especially useful with JOptionPane, where you are building a string for a dialog box:

```
import javax.swing.JOptionPane;

public class FormatDialog {
    public static void main(String[] args) {
        String name = "Randy";
        double balance = 1234.5;

        String msg = String.format("Account: %s%nBalance: $%,.2f", name, balance);
        JOptionPane.showMessageDialog(null, msg);
    }
}
```

```
Account: Randy
Balance: $1,234.50
```

📖 **printf vs String.format**

- **printf()** — formats and prints immediately. Returns void.
- **String.format()** — formats and returns a String you can use later.

Same format specifiers. Different purpose.

6. Type Conversion (Casting)

Java is strongly typed. You cannot freely mix types, but you can convert between them. Two kinds: widening (automatic) and narrowing (manual).

Widening — Automatic

Going from a smaller type to a larger one is always safe — no data is lost. Java handles it automatically.

```
int x = 42;
double d = x;       // int to double, no cast needed
System.out.println(d);  // 42.0

long big = 100L;
double bigD = big;  // long to double, still automatic
```

Narrowing — Manual Cast Required

Going from a larger type to a smaller one can lose data. Java forces you to be explicit with a cast operator: the target type in parentheses.

```
double price = 9.99;
int dollars = (int) price;    // truncates -- drops the decimal
System.out.println(dollars);  // 9, NOT 10 -- casting does NOT round

double gpa = 3.876;
int truncated = (int) gpa;
System.out.println(truncated); // 3
```

> ⚠️ **Casting Truncates — It Does Not Round**
>
> Casting a double to an int drops the decimal portion entirely. `(int) 9.99` gives you 9, not 10. Use `Math.round()` if you need rounding.

Widening Order

```
// byte  short  int  long  float  double
//
// Left to right = widening (automatic)
// Right to left = requires a cast
```

Integer Division Gotcha

Dividing two integers gives you integer division — the decimal is discarded. This surprises beginners constantly.

```
int a = 7;
int b = 2;

System.out.println(a / b);          // 3  -- decimal dropped!
System.out.println((double) a / b); // 3.5 -- cast one operand first
```

> 🔧 **Try It**
>
> What does `(int) 3.999` give you? What about `(int) -3.7`? Run it and check — the behavior on negative numbers surprises students every time.

7. The Math Class

The `Math` class is built into Java — no import needed. Every method is static, so you call them directly: `Math.methodName()`.

Core Math Methods

Method	What It Does	Example	Result
`Math.pow(base, exp)`	Raises base to a power	`Math.pow(2, 10)`	1024.0
`Math.sqrt(x)`	Square root	`Math.sqrt(25)`	5.0
`Math.abs(x)`	Absolute value	`Math.abs(-7)`	7
`Math.round(x)`	Rounds to nearest integer	`Math.round(3.7)`	4
`Math.random()`	Random double 0.0 to <1.0	`Math.random()`	e.g. 0.482...

```
public class MathDemo {
    public static void main(String[] args) {
        System.out.println(Math.pow(3, 4));    // 81.0
        System.out.println(Math.sqrt(144));    // 12.0
        System.out.println(Math.abs(-25));     // 25
        System.out.println(Math.round(4.5));   // 5
        System.out.println(Math.round(4.4));   // 4
    }
}
```

Random Numbers in a Range

`Math.random()` gives you a decimal between 0.0 (inclusive) and 1.0 (exclusive). To get a random integer in a specific range, use this formula:

```
// Random int from min to max (inclusive):
int num = (int)(Math.random() * (max - min + 1)) + min;

// Example: random number from 1 to 6 (simulating a die roll):
int die = (int)(Math.random() * 6) + 1;
System.out.println("You rolled: " + die);
```

```
You rolled: 4
```

> 💡 **Math Methods Return double**
>
> `Math.pow()`, `Math.sqrt()`, and `Math.random()` all return `double`. If you need an `int`, cast the result: `(int) Math.pow(2, 8)` gives you 256.

> 🔧 **Try It**
>
> Write a program that asks the user to enter a number and then prints its square root, its absolute value, and whether it rounds up or down to the nearest integer.

Summary

📋 Module 03 Key Takeaways

- **Scanner** reads different types with different methods: `nextInt()`, `nextDouble()`, `next()`, `nextLine()`, etc.
- **The nextLine() bug** happens when a numeric read leaves a newline in the buffer. Fix it with an extra `input.nextLine();` call.
- **JOptionPane** provides GUI dialogs: `showMessageDialog()` displays, `showInputDialog()` collects. Input always comes back as a String.
- **Integer.parseInt()** and **Double.parseDouble()** convert Strings to numbers.
- **printf()** formats and prints; **String.format()** formats and returns. Same specifiers: `%d`, `%f`, `%s`, `%n`.
- **Widening** (e.g., int to double) is automatic. **Narrowing** (e.g., double to int) requires a cast and truncates.
- **Math class** methods are static: `Math.pow()`, `Math.sqrt()`, `Math.abs()`, `Math.round()`, `Math.random()`.
- Random int in a range: `(int)(Math.random() * (max - min + 1)) + min`

Vocabulary Review

Term	Definition
Scanner	A Java class that reads input from the keyboard (or other sources) one token or line at a time
nextLine()	Scanner method that reads an entire line of input including spaces
next()	Scanner method that reads one word, stopping at whitespace
JOptionPane	A Swing class that provides pop-up dialog boxes for displaying messages and collecting input

Term	Definition
showInputDialog()	JOptionPane method that shows a text-entry dialog and returns the user's input as a String
Integer.parseInt()	Method that converts a String containing an integer value into an actual int
Double.parseDouble()	Method that converts a String containing a decimal value into an actual double
printf()	Output method that uses format specifiers to control exactly how values are displayed
String.format()	Same formatting rules as printf() but returns a String instead of printing
Format specifier	A placeholder in a printf format string (like %d, %f, %s) that gets replaced by a value
Widening conversion	Automatically converting a smaller type to a larger type (e.g., int to double) with no data loss
Narrowing conversion	Manually casting a larger type to a smaller one; may lose data through truncation
Cast operator	The syntax (type) used to force a narrowing conversion, e.g., (int) myDouble
Truncation	Dropping the decimal portion when casting a floating-point number to an integer
Math class	A built-in Java class with static methods for common mathematical operations
Math.random()	Returns a random double value between 0.0 (inclusive) and 1.0 (exclusive)

Knowledge Check

Circle the letter of the best answer for each question.

1. Which Scanner method reads an entire line of text, including spaces?

A) `next()`

B) `nextToken()`

C) `nextLine()`

D) `nextString()`

2. What is the correct way to create a Scanner object to read from the keyboard?

A) `Scanner input = new Scanner(keyboard);`

B) `Scanner input = new Scanner(System.in);`

C) `Scanner input = Scanner.create();`

D) `Scanner input = new Scanner(console);`

3. You call `nextInt()` followed immediately by `nextLine()`. The `nextLine()` returns an empty string. What is the most likely cause?

A) The user pressed Backspace

B) `nextInt()` consumed the newline character

C) `nextLine()` does not work after `nextInt()`

D) The leftover newline from pressing Enter was not consumed

4. Which JOptionPane method displays a message in a pop-up dialog?

A) `showMessageDialog()`

B) `displayMessage()`

C) `popupMessage()`

D) `showDialog()`

5. The user types "25" in a JOptionPane input dialog. To store it as an integer, you write:

A) `int n = Integer.convert(str);`

B) `int n = (int) str;`

C) `int n = str.toInt();`

D) `int n = Integer.parseInt(str);`

6. What does the format specifier `%.2f` do?

A) Prints a float with exactly 2 total digits

B) Prints a decimal number rounded to 2 decimal places

C) Multiplies the value by 2 before printing

D) Prints 2 copies of the floating-point value

7. You want to format a value into a String without printing it yet. Which method should you use?

A) `System.out.printf()`

B) `String.print()`

C) `String.format()`

D) `System.out.format()`

8. What is the result of `(int) 7.9` in Java?

A) 8 (rounds up)

B) 7 (truncates)

C) 8.0

D) A compilation error

9. Which of the following generates a random integer between 1 and 10 inclusive?

A) `(int)(Math.random() * 10)`

B) `Math.random(10)`

C) `(int)(Math.random() * 10) + 1`

D) `(int)(Math.random()) + 10`

10. Which statement about widening type conversion is true?

A) It requires an explicit cast operator

B) It may cause data loss

C) It happens automatically because no data is lost

D) It only works between String types

✅ Answer Key

Question	Answer	Explanation
1	**C**	`nextLine()` reads an entire line of text as a String.
2	**B**	`Scanner keyboard = new Scanner(System.in);` creates a Scanner.
3	**D**	After `nextInt()`, the newline remains in the buffer. Call `nextLine()` to consume it.
4	**A**	`JOptionPane.showMessageDialog(null, "text")` displays a dialog box.
5	**D**	`JOptionPane.showInputDialog()` returns a String; use `Integer.parseInt()` to convert.
6	**B**	`%.2f` formats a floating-point number to exactly 2 decimal places.
7	**C**	`String.format()` returns a formatted String; `printf()` prints directly.
8	**B**	Casting `(int)3.99` truncates to 3 — it does not round.
9	**C**	The formula `(int)(Math.random() * range) + min` generates a random number in range.
10	**C**	Widening conversion (int to double) happens automatically in Java.