Java Programming - I: Session 2 - Variables, Data Types, Operators, and Control Structures

Session Objectives

In this session, we will cover the foundational concepts of Java programming as outlined in Sessions 2 and 3 of *Java Programming - The Complete Guide for Beginners*. By the end, you will be able to:

- Understand variables, their declaration, and naming conventions.
- Explore Java's data types, including primitive and reference types.
- Use escape sequences and format specifiers.
- Identify and apply different types of operators.
- Understand type casting (implicit and explicit).
- Use decision-making constructs (if, switch-case).
- Implement loops (while, do-while, for) and jump statements (break, continue).
- Compare control structures and understand their use cases.

This session builds on the Java development environment set up in Session 1, using Visual Studio Code (VS Code) for coding examples.

1. Variables

What is a Variable?

- A variable is a named storage location in memory that holds a value.
- Purpose: Stores data (e.g., numbers, text) that can be used and modified during program execution.

Syntax of Variable Declaration

```
dataType variableName; // Declaration
variableName = value;   // Initialization
// Or combined:
dataType variableName = value;
```

Example:

```
int age = 25; // Declares an integer variable 'age' and assigns 25
```

Naming Rules and Conventions

- **Rules**:
  - Must start with a letter, underscore (_), or dollar sign ($).
  - Can include letters, digits, underscores, or dollar signs.
  - Case-sensitive (e.g., `Age` and `age` are different).
  - Cannot be a Java keyword (e.g., `class`, `int`).
- **Conventions**:
  - Use camelCase for variable names (e.g., `studentName`).
  - Choose meaningful names (e.g., `totalScore` instead of `ts`).
  - Constants use UPPER_SNAKE_CASE (e.g., `MAX_SCORE`).

Example:

```
double studentGrade = 85.5; // Good naming
int x = 10; // Poor naming, not descriptive
```

2. Data Types

Java has two categories of data types:

- **Primitive Data Types**: Store simple values directly.
- **Reference Data Types**: Store references to objects (e.g., arrays, strings).

Primitive Data Types

Type	Size	Description	Example
`byte`	1 byte	Integer (-128 to 127)	`byte b = 100;`
`short`	2 bytes	Integer (-32,768 to 32,767)	`short s = 1000;`
`int`	4 bytes	Integer (-2^31 to 2^31-1)	`int i = 50000;`
`long`	8 bytes	Integer (-2^63 to 2^63-1)	`long l = 123456L;`
`float`	4 bytes	Floating-point (6-7 digits)	`float f = 3.14f;`
`double`	8 bytes	Floating-point (15 digits)	`double d = 3.14159;
`char`	2 bytes	Single Unicode character	`char c = 'A';`
`boolean`	1 bit	True or false	`boolean b = true;`

Reference Data Types

- Include classes, arrays, interfaces, and strings.
- Example: `String name = "John";` (String is a reference type).
- Store memory addresses pointing to objects, not the data itself.

Escape Sequences

Used in strings to represent special characters:

Sequence	Description
`\n`	New line
`\t`	Tab
`\"`	Double quote
`\\`	Backslash
`\b`	Backspace

Example:

```
System.out.println("Hello\nWorld!"); // Prints on two lines
System.out.println("Path: C:\\Users"); // Prints backslash
```

Format Specifiers

Used with `System.out.printf` for formatted output:

Specifier	Type	Example
`%d`	Integer	`printf("%d", 42);`
`%f`	Floating-point	`printf("%.2f", 3.14159);`
`%s`	String	`printf("%s", "Java");`
`%c`	Character	`printf("%c", 'A');`

Example:

```
int age = 25;
double height = 5.9;
System.out.printf("Age: %d, Height: %.1f\n", age, height);
// Output: Age: 25, Height: 5.9
```

3. Operators

Operators perform operations on variables/values. Java's main operator types:

- **Arithmetic**: `+`, `-`, `*`, `/`, `%` (modulus), `++`, `--`.

  ```
  int a = 10, b = 3;
  System.out.println(a + b); // 13
  System.out.println(a % b); // 1 (remainder)
  System.out.println(++a);   // 11 (increment)
  ```

- **Relational**: `==`, `!=`, `>`, `<`, `>=`, `<=`.

```
System.out.println(a > b); // true
```

- **Logical**: `&&` (AND), `||` (OR), `!` (NOT).

```
boolean x = true, y = false;
System.out.println(x && y); // false
```

- **Assignment**: `=`, `+=`, `-=`, `*=`, `/=`, `%=`.

```
int c = 5;
c += 3; // c = c + 3; now c = 8
```

- **Bitwise**: `&`, `|`, `^`, `~`, `<<`, `>>`.

```
int x = 5; // 0101 in binary
int y = 3; // 0011 in binary
System.out.println(x & y); // 1 (bitwise AND)
```

- **Ternary**: `condition ? valueIfTrue : valueIfFalse`.

```
int max = (a > b) ? a : b; // max = 10
```

4. Type Casting

Converting a value from one data type to another.

- **Implicit (Automatic)**: Smaller type to larger type (no data loss).

```
int i = 100;
double d = i; // Implicit: int to double
System.out.println(d); // 100.0
```

- **Explicit (Manual)**: Larger type to smaller type (may lose data).

```
double d = 3.99;
int i = (int) d; // Explicit: double to int
System.out.println(i); // 3 (truncates decimal)
```

5. Decision-Making Constructs

If Statement

Executes code based on a condition.

- **Forms**:
  - Simple `if`:

```
int age = 18;
if (age >= 18) {
    System.out.println("Adult");
}
```

  - `if-else`:

```
if (age >= 18) {
    System.out.println("Adult");
} else {
    System.out.println("Minor");
}
```

  - `if-else-if`:

```
int score = 85;
if (score >= 90) {
    System.out.println("A");
} else if (score >= 80) {
    System.out.println("B");
} else {
    System.out.println("C");
}
```

Switch-Case Statement

Selects one of many code blocks based on a variable's value.

```
int day = 3;
switch (day) {
    case 1:
        System.out.println("Monday");
        break;
    case 2:
        System.out.println("Tuesday");
        break;
    case 3:
        System.out.println("Wednesday");
        break;
    default:
        System.out.println("Invalid day");
}
```

- **break**: Exits the switch block.
- **default**: Executes if no case matches.

If vs. Switch-Case

Feature	If-Else	Switch-Case
Condition	Any boolean expression	Single value (int, char, String, enum)
Readability	Flexible, but verbose	Cleaner for multiple discrete values
Use Case	Complex conditions	Fixed set of options

6. Loops

Loops execute a block of code repeatedly while a condition is true.

While Loop

- Executes as long as the condition is true.

```java
int i = 1;
while (i <= 5) {
    System.out.println(i);
    i++;
}
// Output: 1 2 3 4 5
```

- **Rule**: Ensure the condition eventually becomes false to avoid infinite loops.

Do-While Loop

- Executes at least once, then checks the condition.

```java
int i = 1;
do {
    System.out.println(i);
    i++;
} while (i <= 5);
// Output: 1 2 3 4 5
```

For Loop

- Ideal for known iteration counts.

```java
for (int i = 1; i <= 5; i++) {
    System.out.println(i);
}
// Output: 1 2 3 4 5
```

- Syntax: `for (initialization; condition; update)`

Nested Loops

- Loops inside loops.

```java
for (int i = 1; i <= 3; i++) {
    for (int j = 1; j <= 2; j++) {
        System.out.println("i=" + i + ", j=" + j);
    }
}
// Output: i=1, j=1
//         i=1, j=2
//         i=2, j=1
//         ...
```

Jump Statements

- **break**: Exits the loop or switch.

```java
for (int i = 1; i <= 10; i++) {
    if (i == 5) break;
    System.out.println(i);
}
// Output: 1 2 3 4
```

- **continue**: Skips the current iteration, continues with the next.

```java
for (int i = 1; i <= 5; i++) {
    if (i == 3) continue;
    System.out.println(i);
}
// Output: 1 2 4 5
```

Comparing Loops

Loop Type	When to Use	Example Use Case
`while`	Unknown number of iterations	Reading input until EOF
`do-while`	At least one execution needed	Menu-driven program
`for`	Known number of iterations	Iterating over an array

7. Coding in VS Code

- Open your Java project in VS Code (from Session 1 setup).
- Create a new file, e.g., `Session2Demo.java`, and try this example:

```
public class Session2Demo {
    public static void main(String[] args) {
        // Variables and Data Types
        int age = 20;
        double height = 5.9;
        String name = "Alice";
        System.out.printf("Name: %s, Age: %d, Height: %.1f\n", name, age, height);

        // Operators
        int a = 10, b = 3;
        System.out.println("Sum: " + (a + b));
        System.out.println("Modulus: " + (a % b));

        // If-Else
        if (age >= 18) {
            System.out.println(name + " is an adult.");
        } else {
            System.out.println(name + " is a minor.");
        }

        // For Loop
        for (int i = 1; i <= 5; i++) {
            if (i == 3) continue;
            System.out.println("Number: " + i);
        }
    }
}
```

- **Run**: Right-click the file in VS Code and select **Run Java**, or use the terminal:

```
javac Session2Demo.java
java Session2Demo
```

8. Next Steps

- Review Sessions 2 and 3 in *Java Programming - The Complete Guide for Beginners*.
- Prepare for Session 3, which covers "Try It Yourself" questions for Sessions 1–3.
- Practice coding exercises in VS Code to reinforce concepts.
- Access OnlineVarsity for eBooks, practice tests, and lab assignments.

Additional Resources

- *Head First Java* by Kathy Sierra & Bert Bates.
- *Java: A Beginner's Guide* by Herbert Schildt.
- Oracle Java Tutorials: https://docs.oracle.com/javase/tutorial/java/nutsandbolts/.
- VS Code Java Guide: https://code.visualstudio.com/docs/languages/java.