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

Session Objectives

This session covers Sessions 2 and 3 from *Java Programming - The Complete Guide for Beginners*. By the end, you will:

- Declare and initialize variables, following naming conventions.
- Understand Java's primitive and reference data types.
- Use escape sequences and format specifiers.
- Apply arithmetic, relational, logical, and other operators.
- Perform implicit and explicit type casting.
- Implement decision-making constructs (if, switch-case).
- Use loops (while, do-while, for) and jump statements (break, continue).
- Compare control structures for appropriate use cases.

All coding will be done in Visual Studio Code (VS Code), building on the environment setup from Session 1.

Sub-Session 1: Variables

Definition

A variable is a named memory location that stores a value of a specific data type, which can be used and modified during program execution.

Explanation

- Variables act as containers for data (e.g., numbers, text).
- Declaration specifies the data type and name; initialization assigns a value.
- **Syntax**:

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

- **Naming Rules**:
  - Start with a letter, underscore (_), or dollar sign ($).
  - Include letters, digits, underscores, or dollar signs.
  - Case-sensitive; cannot use Java keywords (e.g., `int`, `class`).
- **Naming Conventions**:
  - Use camelCase (e.g., `studentAge`).
  - Choose descriptive names (e.g., `totalMarks` vs. `tm`).
  - Constants use UPPER_SNAKE_CASE (e.g., `MAX_SCORE`).

Examples

1. Declaring and initializing variables:

```
int age = 25;
double salary = 45000.50;
String name = "Alice";
```

2. Invalid variable names:

```
int 2count; // Invalid: starts with a number
int class;  // Invalid: uses a keyword
```

Class Work

1. Declare an `int` variable `score` and initialize it to 90.
2. Declare a `String` variable `studentName` and set it to your name.
3. Write a program to print the values of `score` and `studentName` using `System.out.println`.

Sub-Session 2: Data Types

Definition

Data types specify the type and size of data a variable can hold, categorized as primitive (simple values) or reference (objects).

Explanation

- **Primitive Data Types**:
  - Store basic values directly.
  - Include: `byte` , `short` , `int` , `long` , `float` , `double` , `char` , `boolean` .
  - Example: `int` for whole numbers, `double` for decimals.
- **Reference Data Types**:
  - Store memory addresses of objects (e.g., `String` , arrays, classes).
  - Example: `String name = "John";` references a String object.
- **Key Primitive Types**:

Type	Size	Range/Example
`byte`	1 byte	-128 to 127 (`byte b = 100;`)
`int`	4 bytes	-2^31 to 2^31-1 (`int i = 500;`)
`double`	8 bytes	15-digit precision (`double d = 3.14;`)
`char`	2 bytes	Unicode char (`char c = 'A';`)
`boolean`	1 bit	`true` / `false` (`boolean b = true;`)

Examples

1. Primitive types:

```java
int count = 10;
double price = 19.99;
char grade = 'A';
boolean isActive = true;
```

2. Reference type:

```java
String message = "Welcome to Java!";
```

Class Work

1. Declare a `double` variable `temperature` and set it to 23.5.
2. Declare a `boolean` variable `isStudent` and set it to `true` .
3. Write a program to print both variables.

Sub-Session 3: Escape Sequences and Format Specifiers

Definition

- **Escape Sequences**: Special characters in strings, prefixed with a backslash (`\`), to represent non-printable characters.
- **Format Specifiers**: Placeholders in `System.out.printf` for formatting output.

Explanation

- **Escape Sequences**:
  - Used in strings to insert special characters.
  - Common sequences: `\n` (new line), `\t` (tab), `\"` (quote), `\\` (backslash).
- **Format Specifiers**:
  - Used with `printf` to format output.
  - Examples: `%d` (integer), `%f` (floating-point), `%s` (string), `%c` (char).

Examples

1. Escape sequences:

   ```
   System.out.println("Line 1\nLine 2"); // New line
   System.out.println("Path: C:\\Users"); // Backslash
   ```

   Output:

   ```
   Line 1
   Line 2
   Path: C:\Users
   ```

2. Format specifiers:

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

   Output: `Age: 25, Height: 5.9`

Class Work

1. Write a string with a tab between "Java" and "Programming".
2. Use `printf` to print a student's name and score with format specifiers.
3. Create a program that prints a path like `D:\Data\file.txt` using an escape sequence.

Sub-Session 4: Operators

Definition

Operators are symbols that perform operations on variables/values, such as arithmetic, comparison, or logical operations.

Explanation

- **Arithmetic**: `+`, `-`, `*`, `/`, `%` (modulus), `++`, `--`.
- **Relational**: `==`, `!=`, `>`, `<`, `>=`, `<=`.
- **Logical**: `&&` (AND), `||` (OR), `!` (NOT).
- **Assignment**: `=`, `+=`, `-=`, `*=`, `/=`, `%=`.
- **Bitwise**: `&`, `|`, `^`, `~`, `<<`, `>>` (advanced, used for bit manipulation).
- **Ternary**: `condition ? valueIfTrue : valueIfFalse`.

Examples

1. Arithmetic and Assignment:

```java
int a = 10, b = 3;
System.out.println(a + b); // 13
System.out.println(a % b); // 1
a += 5; // a = a + 5; now a = 15
```

2. Relational and Logical:

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

3. Ternary:

```java
int max = (a > b) ? a : b; // max = 15
```

Class Work

1. Write a program to calculate the product of two numbers using `*`.
2. Use a relational operator to check if a number is positive.
3. Create a program using the ternary operator to find the smaller of two numbers.

Sub-Session 5: Type Casting

Definition

Type casting is the process of converting a value from one data type to another, either implicitly (automatic) or explicitly (manual).

Explanation

- **Implicit Casting**: Automatic conversion from smaller to larger type (no data loss).
  - Example: `int` to `double`.
- **Explicit Casting**: Manual conversion from larger to smaller type (may lose data).
  - Syntax: `(targetType) value`
  - Example: `double` to `int` (truncates decimal).

Examples

1. Implicit:

   ```
   int i = 100;
   double d = i; // d = 100.0
   ```

2. Explicit:

   ```
   double d = 3.99;
   int i = (int) d; // i = 3
   ```

Class Work

1. Convert an `int` (50) to a `double` and print it.
2. Convert a `double` (99.99) to an `int` and print the result.
3. Write a program that demonstrates both implicit and explicit casting.

Sub-Session 6: Decision-Making Constructs

Definition

Decision-making constructs (`if` , `switch-case`) execute code based on conditions.

Explanation

- **If Statement**:
  - Executes if a condition is true.
  - Forms: `if` , `if-else` , `if-else-if` .
- **Switch-Case**:
  - Selects a code block based on a variable’s value.
  - Uses `break` to exit; `default` for unmatched cases.
- **If vs. Switch**:
  - `if` : Flexible for complex conditions.
  - `switch` : Cleaner for discrete values (e.g., `int` , `String`).

Examples

1. If-Else:

```java
int age = 16;
if (age >= 18) {
    System.out.println("Adult");
} else {
    System.out.println("Minor");
}
```

2. Switch-Case:

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

Class Work

1. Write an `if-else` program to check if a number is even or odd.
2. Create a `switch-case` program to print the day name for a number (1–7).
3. Write a program using `if-else-if` to assign grades (A, B, C) based on a score.

Sub-Session 7: Loops

Definition

Loops (`while` , `do-while` , `for`) execute a block of code repeatedly while a condition is true.

Explanation

- **While Loop**: Runs while condition is true; may not execute if condition is initially false.
- **Do-While Loop**: Runs at least once, then checks condition.
- **For Loop**: Ideal for known iteration counts; includes initialization, condition, and update.
- **Nested Loops**: Loops within loops for complex iterations.
- **Jump Statements**:
  - `break` : Exits the loop or switch.
  - `continue` : Skips current iteration, proceeds to next.
- **Loop Comparison**:

Loop	Use Case
`while`	Unknown iterations
`do-while`	At least one execution
`for`	Known iterations

Examples

1. While:

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

2. Do-While:

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

3. For with continue:

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

Class Work

1. Write a `for` loop to print numbers 10 to 1 (descending).
2. Create a `while` loop to print even numbers from 2 to 10.
3. Write a program with a nested loop to print a 3x3 grid of numbers.

General Objective Questions

1. What is the purpose of a variable in Java?
   - a) To store methods
   - b) To store data values
   - c) To define classes
   - d) To execute loops

   Answer: b) To store data values
2. Which data type is used for decimal numbers with high precision?
   - a) `int`
   - b) `float`
   - c) `double`
   - d) `char`

   Answer: c) `double`
3. What does the escape sequence `\n` do?
   - a) Adds a tab
   - b) Starts a new line

- c) Prints a backslash
- d) Adds a quote

  Answer: b) Starts a new line

4. Which operator checks for equality between two values?
   - a) `=`
   - b) `==`
   - c) `!=`
   - d) `>=`

   Answer: b) `==`
5. When should you use a `do-while` loop instead of a `while` loop?
   - a) When the loop should never execute
   - b) When the loop should execute at least once
   - c) When the number of iterations is fixed
   - d) When nested loops are needed

   Answer: b) When the loop should execute at least once

Practical Questions

1. Write a program in VS Code that:
   - Declares an `int` variable `marks` and a `String` variable `grade`.
   - Uses `if-else-if` to assign `grade` as "A" (marks >= 90), "B" (marks >= 80), or "C" (marks < 80).
   - Prints the result using `printf`.
2. Create a program that:
   - Uses a `for` loop to print the first 10 multiples of 5.
   - Skips the 5th multiple using `continue`.
3. Write a program that:
   - Uses a `switch-case` statement to convert a number (1–12) to a month name (e.g., 1 = January).
   - Includes a `default` case for invalid inputs.

Instructions for Coding

- Open VS Code and create a new file (e.g., `Session2Practice.java`).
- Write and test your code using **Run Java** (right-click) or the terminal:

```
javac Session2Practice.java
java Session2Practice
```

- Use the debugger (`F5`) to step through code if needed.

Next Steps

- Review Sessions 2 and 3 in *Java Programming - The Complete Guide for Beginners*.
- Prepare for Session 3, covering "Try It Yourself" questions for Sessions 1–3.
- 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.