Module 1: Java Basics

Premanand S

Assistant Professor
School of Electronics Engineering (SENSE)
Vellore Instittute of Technology
Chennai Campus

premanand.s@vit.ac.in

January 4, 2025

Topics covered in Module 1,

- OOP Paradigm
- Features of JAVA Language
- JVM
- Bytecode
- Java Program Structure
- Basic Programming Construct
- Data Types
- Variables
- Java naming conventions
- Operators

Java Program Structure

Java Program Structure

Overview: Java programs follow a specific structure similar to how a book is organized.

- **1. Documentation Section (Comments)**
  Describes what the program does.
  - `// This is a simple Java program`
  - `/* Multi-line comment */`
- **2. Package Declaration (Optional)**
  Groups related classes.
  - `package mypackage;`
- **3. Import Statements (Optional)**
  Imports built-in or user-defined classes.
  - `import java.util.Scanner;`
- **4. Class Declaration**
  Defines a class.
  - `public class Main { ... }`

Java Program Structure

- **5. Main Method Declaration**
  The entry point where the program starts.
  - `public static void main(String[] args) { ... }`
- **6. Variable Declarations (Optional)**
  Declares variables.
  - `int number = 10;`
  - `String name = "Java";`
- **7. Method Definition (Optional)**
  Defines functions.
  - `public static void greet() { System.out.println("Welcome to Java!"); }`
- **8. Statements and Logic**
  Instructions to be executed.
  - `System.out.println("Hello, World!");`

Complete Java Program Example

```
// Documentation Section
/* This program prints "Hello, World!" */

import java.util.Scanner;

// Class Declaration
public class Main {

    // Main Method Declaration
    public static void main(String[] args) {
        // Variable Declaration
        String greeting = "Hello, World!";

        // Method Call
        printMessage(greeting);
    }
```

Complete Java Program Example (Contd...)

```
    // Method Definition
    public static void printMessage(String message) {
        System.out.println(message);  // Statement to print
    }
}
```

Summary of Java Program Structure

Section	**Purpose**
Documentation Section	Describes the program using comments
Package Declaration	Organizes classes into packages
Import Statements	Imports built-in or user-defined classes
Class Declaration	Defines the program's structure
Main Method Declaration	Starting point for program execution
Variable Declarations	Defines variables
Method Definition	Contains reusable code blocks
Statements and Logic	Contains the program logic

Basic Programming Construct

Basic Programming Constructs

Overview: Programming constructs are the building blocks of any programming language, helping in creating efficient programs.

- **1. Sequential Constructs**
  Instructions are executed in the order they appear.
  - Example: `int x = 10; System.out.println(x);`
- **2. Selection Constructs**
  Decisions are made based on conditions.
  - Example: `if (x > 0) { System.out.println("Positive"); }`
- **3. Iteration Constructs**
  Repeating instructions until a condition is met.
  - Example: `for (int i = 0; i < 5; i++) { System.out.println(i); }`
- **4. Function/Procedure Calls**
  Reusable blocks of code that perform specific tasks.
  - Example: `public static void greet() { System.out.println("Hello!"); }`

Complete Example: Basic Constructs in Java

Example (Understanding)

```
// Sequential
int x = 10;
System.out.println(x);

// Selection
if (x > 0) {
    System.out.println("Positive");
}

// Iteration
for (int i = 0; i < 5; i++) {
    System.out.println(i);
}
```

Complete Example: Basic Constructs in Java (Contd...)

Example (Understanding)

```
// Function/Procedure Call
public static void greet() {
    System.out.println("Hello!");
}
```

Data Types

Data Types in Programming

Overview: Data types define the type of data a variable can hold. They help manage memory efficiently and ensure data integrity.

- **1. Primitive Data Types**
  Basic types provided by the language.
  - `int` - Integer numbers (e.g., 10, -5)
  - `float` - Floating-point numbers (e.g., 3.14, -0.99)
  - `char` - Single characters (e.g., 'A', 'b')
  - `boolean` - True/False values
- **2. Derived Data Types**
  Created from primitive types.
  - `arrays` - Collection of similar elements
  - `classes` - User-defined types with attributes and methods
- **3. Abstract Data Types**
  Defined by their behavior.
  - `List, Stack, Queue` - Collections with specific operations
- **4. User-Defined Data Types**
  Custom types defined by users.
  - `struct, enum` - Custom groupings and enumerations

Integer Data Types in Java

In Java, integer data types are used to store whole numbers. There are four main integer types:

- **byte**:
  - Size: 1 byte (8 bits)
  - Range: -128 to 127
  - Example: `byte a = 100;`
- **short**:
  - Size: 2 bytes (16 bits)
  - Range: -32,768 to 32,767
  - Example: `short b = 25000;`
- **int**:
  - Size: 4 bytes (32 bits)
  - Range: -2^{31} to $2^{31}-1$ (approximately -2 billion to 2 billion)
  - Example: `int c = 100000;`
- **long**:
  - Size: 8 bytes (64 bits)
  - Range: -2^{63} to $2^{63}-1$ (very large range)
  - Example: `long d = 10000000000L;`

Integer data type

Example (Java)

```
public class IntegerTypesExample {
    public static void main(String[] args) {
        byte byteValue = 100;
        short shortValue = 25000;
        int intValue = 100000;
        long longValue = 10000000000L;

        System.out.println("Byte value: " + byteValue);
        System.out.println("Short value: " + shortValue);
        System.out.println("Int value: " + intValue);
        System.out.println("Long value: " + longValue);
    }
}
```

Float Data Type in Java

In Java, the `float` data type is used to store floating-point numbers (i.e., numbers with decimal points).

- **float**:
  - Size: 4 bytes (32 bits)
  - Range: $\pm$ 1.4E-45 to $\pm$ 3.4E+38
  - Precision: 6 to 7 decimal places
  - Example: `float pi = 3.14f;`
  - Use f or F to indicate a float literal

Float

Example (Understanding)

```
public class FloatExample {
    public static void main(String[] args) {
        float pi = 3.14f;
        float temperature = 36.6f;
        System.out.println("Value of pi: " + pi);
        System.out.println("Temperature: " + temperature);
    }
}
```

Double Data Type in Java

In Java, the `double` data type is used to store double-precision floating-point numbers (i.e., numbers with decimal points).

- **double**:
  - Size: 8 bytes (64 bits)
  - Range: $\pm$ 4.9E-324 to $\pm$ 1.8E+308
  - Precision: 15 to 16 decimal places
  - Example: `double pi = 3.141592653589793;`
  - `double` provides higher precision compared to `float`.

Double

Example (Understanding)

```
public class DoubleExample {
    public static void main(String[] args) {
        double pi = 3.141592653589793;
        // Speed of light in m/s
        double distance = 299792458.0;
        System.out.println("Value of pi: " + pi);
        System.out.println("Speed of light: " + distance
                                + " m/s");
    }
}
```

Char Data Type in Java

In Java, the `char` data type is used to store single characters or Unicode values.

- **char**:
  - Size: 2 bytes (16 bits)
  - Range: 0 to 65,535 (0 to `''` in Unicode)
  - Can store a single character or a Unicode value.
  - Example: `char letter = 'A';`
  - Unicode representation allows storing characters from various languages and symbols.

Char

Example (Understanding)

```
public class CharExample {
    public static void main(String[] args) {
        char letter = 'A';
        char symbol = '$';
        System.out.println("Letter: " + letter);
        System.out.println("Symbol: " + symbol);
    }
}
```

String

- Definition: A sequence of characters stored as a reference type.
- The .length() method in Java is used to determine the number of characters in a string. It returns an integer value representing the length of the string.

Example (Understanding)

```
public class StringExample {
    public static void main(String[] args) {
        String name = "Java Programming";
        System.out.println("Name: " + name);
        System.out.println("Length: " + name.length());
    }
}
```

Boolean Data Type in Java

In Java, the `boolean` data type is used to store true or false values.

- **boolean**:
    - Size: 1 bit (although JVM may allocate 1 byte for practical purposes)
    - Values: `true` or `false`
    - Typically used for logical operations and conditions.
    - Example: `boolean isActive = true;`

Boolean

Example (Understanding)

```
public class BooleanExample {
    public static void main(String[] args) {
        boolean isActive = true;
        boolean isAdult = false;
        System.out.println("Is Active: " + isActive);
        System.out.println("Is Adult: " + isAdult);
    }
}
```

Arrays in Java

In Java, an `array` is a collection of elements of the same type stored in contiguous memory locations.

- **Array Declaration and Initialization**:
  - Syntax: `type[] arrayName;` or `type arrayName[];`
  - Example: `int[] numbers;` or `int numbers[];`
- **Array Initialization**:
  - Syntax: `arrayName = new type[size];`
  - Example: `numbers = new int[5];`
- **Array Initialization with Values**:
  - Example: `int[] numbers = {1, 2, 3, 4, 5};`

Arrays

Example (Understanding)

```
public class ArrayExample {
    public static void main(String[] args) {
    // Array initialization with values
        int[] numbers = {1, 2, 3, 4, 5};

        // Accessing elements
        System.out.println("First element: " + numbers[0]);
        System.out.println("Third element: " + numbers[2]);

        // Modifying an element
        numbers[2] = 10;
        System.out.println("Modified third element: "
                                     + numbers[2]);
```

Arrays (Contd...)

Example (Understanding)

```
        // Loop through array
        System.out.println("All elements in the array:");
        for (int i = 0; i < numbers.length; i++) {
            System.out.println(numbers[i]);
        }
    }
}
```

Classes in Java

In Java, a `class` is a blueprint for creating objects. It defines a data structure that includes fields (variables) and methods (functions) to operate on that data.

- **Class Declaration**:
  - Syntax:

    `void drive()  System.out.println("Driving the car");`
- **Constructor**:
  - A special method used to initialize objects.
  - Same name as the class.
  - No return type.
- **Object Creation**:
  - Syntax: `ClassName objectName = new ClassName();`
  - Example: `Car myCar = new Car();`

Class

Example (Understanding)

```
public class Car {
    String model;
    int year;

    // Constructor
    public Car(String model, int year) {
        this.model = model;
        this.year = year;
    }

    // Method
    public void drive() {
        System.out.println("Driving the " + model + " car");
    }
```

Class (Contd...)

Example (Understanding)

```
    public static void main(String[] args) {
        // Creating an object of Car
        Car myCar = new Car("Tesla", 2023);
        myCar.drive(); // Calling the method
    }
}
```

Type Casting in Java

Type Casting in Java refers to converting a variable from one data type to another.

- **Why Use Type Casting?** - To perform data conversions. - To utilize memory efficiently. - To work with different types in operations.

Types of Type Casting in Java

1. **Widening Casting (Automatic/Implicit)** - Converts a smaller type to a larger type automatically. - No data loss. - Example: `int` to `float`.
2. **Narrowing Casting (Manual/Explicit)** - Converts a larger type to a smaller type manually. - May cause data loss. - Requires explicit casting using parentheses. - Example: `double` to `int`.

Widening Casting Example

- Implicit Casting (Widening): Automatic conversion (e.g., int to float).
- Explicit Casting (Narrowing): Manual conversion (e.g., float to int).

Example (Understanding)

```
public class WideningExample {
    public static void main(String[] args) {
        int num = 100;
        double result = num; // Automatic Casting
        System.out.println("Widened Value: " + result);
    }
}
\textbf{Output:}
Widened Value: 100.0
```

Narrowing Casting Example

- Implicit Casting (Widening): Automatic conversion (e.g., int to float).
- Explicit Casting (Narrowing): Manual conversion (e.g., float to int).

Example (Understanding)

```
public class NarrowingExample {
    public static void main(String[] args) {
        double num = 99.99;
        int result = (int) num; // Explicit Casting
        System.out.println("Narrowed Value: " + result);
    }
}
\textbf{Output:}
Narrowed Value: 99
```

Variables

Variables in Java

Definition: A variable in Java is a container that holds data during the execution of a program. Each variable has a data type and a unique name.

- **Why Use Variables?**
  - Store values temporarily.
  - Make programs dynamic and reusable.
  - Facilitate data manipulation.

Types of Variables in Java

1. **Local Variables**
   - Declared inside a method or block.
   - Accessible only within that method or block.
   - No default value; must be initialized.
2. **Instance Variables (Non-static)**
   - Declared inside a class but outside methods.
   - Unique to each object (non-static).
   - Default values: 0, `false`, `null`.
3. **Static Variables (Class Variables)**
   - Declared with the `static` keyword.
   - Shared among all objects of the class.
   - Initialized only once when the class is loaded.

Declaring Variables in Java

Syntax: `dataType variableName = value;`

Example (Understanding)

```
int age = 25;             // Integer variable
float price = 12.99f;     // Floating-point variable
char grade = 'A';         // Character variable
boolean isActive = true;// Boolean variable
```

Example Program: Using Variables

Syntax: `dataType variableName = value;`

Example (Understanding)

```
public class VariableExample {
// Static variable
    static String companyName = "TechCorp";
    public static void main(String[] args) {
        int age = 30; // Local variable
        VariableExample obj = new VariableExample();
        obj.displayInfo(age);
    }
    void displayInfo(int age) {
        String name = "Java";  // Instance variable
        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
        System.out.println("Company: " + companyName);   } }
```

Java naming conventions

Java Naming Conventions

Definition: Naming conventions in Java are guidelines for naming variables, methods, classes, and other elements. Following these conventions ensures code readability, maintainability, and consistency.

Why Follow Naming Conventions?

- Improves code readability.
- Reduces maintenance time.
- Enables team collaboration.
- Ensures consistency across projects.

General Naming Rules in Java

- Use meaningful and descriptive names.
- No spaces or special characters except _ and $.
- Names should not start with a digit.
- Use camelCase for methods and variables.
- Use PascalCase for class names.
- Constants should be written in `UPPER_CASE`.

Java Naming Conventions by Type

- **Classes:** Use **PascalCase** (each word starts with a capital letter). Example: `EmployeeDetails`, `StudentInfo`.
- **Methods:** Use **camelCase** (first word lowercase, subsequent words capitalized). Example: `calculateSalary()`, `getDetails()`.
- **Variables:** Use **camelCase**. Example: `employeeName`, `totalMarks`.
- **Constants:** Use **UPPER_CASE_WITH_UNDERSCORES**. Example: `MAX_VALUE`, `PI`.
- **Packages:** Use all lowercase. Example: `com.example.app`.

Example: Java Naming Conventions

Example (Understanding)

```
public class EmployeeDetails {

    // Constant
    public static final int MAX_AGE = 60;

    // Instance variable
    private String employeeName;

    // Constructor
    public EmployeeDetails(String employeeName) {
        this.employeeName = employeeName;
    }
```

Example: Java Naming Conventions (Contd...)

Example (Understanding)

```
    // Method
    public void displayInfo() {
        System.out.println("Employee Name: " + employeeName);
    }

    // Main method
    public static void main(String[] args) {
        EmployeeDetails emp = new EmployeeDetails("Prem");
        emp.displayInfo();
    }
}
```

Operators

Operators in Java

Definition: Operators in Java are special symbols or keywords that perform operations on variables and values. They are used to perform arithmetic, logical, relational, and other computations.

Types of Operators in Java

- **Arithmetic Operators**
- **Relational (Comparison) Operators**
- **Logical Operators**
- **Bitwise Operators**
- **Assignment Operators**
- **Unary Operators**
- **Ternary Operator**

Arithmetic Operators

Description: Used for basic mathematical calculations.

Operator	Description	Example
+	Addition	`a + b`
-	Subtraction	`a - b`
*	Multiplication	`a * b`
/	Division	`a / b`
%	Modulus	`a % b`

Examples of Arithmetic Operators

Example (Java Code)

```
# Example 1: Addition
int a = 10, b = 5;
int sum = a + b;
System.out.println("Sum: " + sum);

# Example 2: Subtraction
int a = 10, b = 5;
int difference = a - b;
System.out.println("Difference: " + difference);
```

Examples of Arithmetic Operators (Contd.)

Example (Java Code)

```
# Example 3: Multiplication
int a = 10, b = 5;
int product = a * b;
System.out.println("Product: " + product);

# Example 4: Integer Division
int a = 10, b = 3;
int quotient = a / b;
System.out.println("Quotient: " + quotient);
```

Examples of Arithmetic Operators (Contd.)

Example (Java Code)

```
# Example 5: Modulus
int a = 10, b = 3;
int remainder = a % b;
System.out.println("Remainder: " + remainder);

# Example 4: Operator Precedence
int result = 10 + 5 * 2; // result = 20, not 30
```

Examples of Arithmetic Operators (Contd.)

- Write a program to calculate the area of a rectangle using $*$ (length $\times$ width).
- Write a program to divide two numbers and print both the quotient and remainder.
- Write a program to check if a number is divisible by 5 and 3 using the modulus operator.
- Write a program to calculate the sum of the first 10 natural numbers.
- Evaluate the following expression and explain the result:

Example (Expression)

```
int result = 10 - 5 * 3 + 4 / 2 % 3;
```

Examples of Arithmetic Operators (Contd.)

- Write a program to calculate the area of a rectangle using $*$ (length $\times$ width).

Example (Java)

```
public class Main {
    public static void main(String[] args) {
        int length = 10;
        int width = 5;

        // Calculate the area of the rectangle
        int area = length * width;

        // Display the result
        System.out.println("The area of the rectangle is:
                                                " + area);
    }
}
```

Examples of Arithmetic Operators (Contd.)

- Write a program to divide two numbers and print both the quotient and remainder.

Example (Java)

```java
public class Main {
    public static void main(String[] args) {
        int dividend = 10;
        int divisor = 3;

        // Calculate the quotient and remainder
        int quotient = dividend / divisor;
        int remainder = dividend % divisor;

        // Print the results
        System.out.println("Quotient: " + quotient);
        System.out.println("Remainder: " + remainder);
    }}
```

Examples of Arithmetic Operators (Contd.)

- Write a program to check if a number is divisible by 5 and 3 using the modulus operator.

Example (Java)

```
public class Main {
    public static void main(String[] args) {
        int number = 15; // Example number

        // Print the result of the divisibility check
        System.out.println(number + " is divisible by
        both 5 and 3: " +
        (number % 5 == 0 && number % 3 == 0));
    }
}
```

Examples of Arithmetic Operators (Contd.)

- Write a program to calculate the sum of the first 10 natural numbers

Example (Expression)

```
public class Main {
    public static void main(String[] args) {
        int sum = 0; // Initialize sum to 0

        // Loop through the first 10 natural numbers
        for (int i = 1; i <= 10; i++) {
            sum += i; // Add the current number to the sum
        }

        // Print the result
        System.out.println("The sum of the first 10 natural
        numbers is: " + sum);
    }
}
```

Examples of Arithmetic Operators (Contd.)

- Evaluate the following expression and explain the result:

Example (Expression)

```
int result = 10 - 5 * 3 + 4 / 2 % 3;

public class Main {
    public static void main(String[] args) {
        // Declare and initialize the result variable
        int result = 10 - 5 * 3 + 4 / 2 % 3;

        // Output the result
        System.out.println("The result of the expression
        is: " + result);
    }
}
```

Operator Precedence in Java

- Parentheses ()
- Multiplication $*$, Division $/$, and Modulus % (from left to right)
- Addition + and Subtraction − (from left to right)