Module 1: An Overview of Java - Java OOP Fundamentals

This document provides an overview of Java, focusing on its object-oriented programming paradigm, basic syntax, data types, operators, and control statements. It is based on the NDU Curriculum DCSA250059.

---

1. Object-Oriented Programming Paradigm

What is a Programming Paradigm?

A programming paradigm is a fundamental style or way of programming that guides how we structure and write code.

Procedural vs Object-Oriented Programming

Feature	Procedural Programming	Object-Oriented Programming
Program Division	Divided into functions/procedures	Divided into objects
Data Security	Low (data is global)	High (data hiding via encapsulation)
Approach	Top-down approach	Bottom-up approach
Data Movement	Data moves freely between functions	Data is encapsulated within objects
Reusability	Limited	High (inheritance, polymorphism)
Real-world Modeling	Difficult	Natural and intuitive
Examples	C, Pascal, FORTRAN	Java, C++, Python, C#

Core OOP Concepts (The Four Pillars)

1. Encapsulation

Encapsulation is the mechanism of wrapping data (variables) and code (methods) together as a single unit. It also involves hiding the internal state of an object from the outside world and only exposing a public interface to interact with it.

```
class BankAccount {
```

```
    private double balance;  // hidden data

    public void deposit(double amount) {  // public interface
        if (amount > 0) {
            balance += amount;
        }
    }

    public double getBalance() {
        return balance;
    }
```

```
}
```

2. Inheritance

Inheritance is a mechanism in which one class acquires the properties and behaviors (fields and methods) of another class. This promotes code reusability.

```
class Animal {
```

```
    void eat() { System.out.println("Eating..."); }
}

class Dog extends Animal {  // Dog inherits Animal
    void bark() { System.out.println("Barking..."); }
```

```
}
```

3. Polymorphism

Polymorphism means
one name, many forms. It allows objects of different classes to be treated as objects of a common type. Java supports two main types of polymorphism:

- **Compile-time polymorphism (Method Overloading)**: Achieved when there are multiple methods with the same name but different parameter lists within the same class.
- **Runtime polymorphism (Method Overriding)**: Achieved when a subclass provides a specific implementation for a method that is already defined in its superclass.

```
// Compile-time polymorphism (Overloading)
class Calculator {
    int add(int a, int b) { return a + b; }
    double add(double a, double b) { return a + b; }
}

// Runtime polymorphism (Overriding)
class Animal {
    void sound() { System.out.println("Animal makes sound"); }
}
class Cat extends Animal {
    void sound() { System.out.println("Meow"); }  // Overrides
}
```

4. Abstraction

Abstraction is the concept of hiding the complex implementation details and showing only the essential features of the object. It focuses on *what* an object does rather than *how* it does it.

```
abstract class Vehicle {
    abstract void start();  // What it does (not how)
}

class Car extends Vehicle {
    void start() {  // How it's done
        System.out.println("Car starts with key");
    }
```

```
}
```

---

2. First Simple Java Program

Hello World Program

```
// File: HelloWorld.java
```

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
```

```
}
```

Program Explanation

```
public class HelloWorld {
```

```
// └─┬─┘ └─┬─┘ └──┬──┘
//   │     │       └── Class name (must match filename)
//   │     └── Class keyword
//   └── Access modifier (accessible everywhere)

    public static void main(String[] args) {
    // └─┬─┘ └─┬─┘ └┬┘ └──┬──┘ └─┬─┘
    //   │     │     │     │       └── Command line parameters
    //   │     │     │     └── Method name (entry point)
    //   │     │     └── Return type (void = nothing)
    //   │     └── Belongs to class, not object
    //   └── Accessible everywhere

        System.out.println("Hello, World!");
        // └──┬──┘ └──┬──┘
        //   │       └── Print with new line
        //   └── Standard output stream
```

```
    }
}
```

Compilation and Execution

```
# Compile - creates HelloWorld.class (bytecode)
javac HelloWorld.java

# Execute - runs on JVM
java HelloWorld

# Output:
Hello, World!
```

Another Simple Program

```
public class SimpleProgram {
    public static void main(String[] args) {
        // Variable declaration and initialization
        int number = 10;
        String message = "Welcome to Java";

        // Printing
        System.out.println("Number: " + number);
        System.out.println("Message: " + message);

        // Arithmetic operation
        int sum = number + 5;
        System.out.println("Sum: " + sum);
    }
}
```

Output:

```
Number: 10
```

```
Message: Welcome to Java
```

```
Sum: 15
```

---

3. Second Short Program

```
public class SecondProgram {
```

```
    public static void main(String[] args) {
        // Taking input from command line arguments
        if (args.length > 0) {
            System.out.println("Hello, " + args[0] + "!");
        } else {
            System.out.println("Hello, User!");
        }

        // Using command line arguments as numbers
        if (args.length >= 2) {
            int a = Integer.parseInt(args[0]);
            int b = Integer.parseInt(args[1]);
            System.out.println("Sum: " + (a + b));
        }
    }
```

```
}
```

Run:

```
java SecondProgram Alice 10 20
```

Output:

```
Hello, Alice!
```

```
Sum: 30
```

---

4. Two Control Statements

if-else Statement

```
public class IfElseDemo {
```

```
    public static void main(String[] args) {
        int marks = 85;

        if (marks >= 90) {
            System.out.println("Grade: A");
        } else if (marks >= 80) {
            System.out.println("Grade: B");
        } else if (marks >= 70) {
            System.out.println("Grade: C");
        } else if (marks >= 60) {
            System.out.println("Grade: D");
        } else {
            System.out.println("Grade: F");
        }

        // Ternary operator (short form of if-else)
        int age = 20;
        String status = (age >= 18) ? "Adult" : "Minor";
        System.out.println("Status: " + status);
    }
```

```
}
```

for Loop

```
public class ForLoopDemo {
```

```
    public static void main(String[] args) {
        // Print numbers 1 to 10
        System.out.println("Numbers 1 to 10:");
        for (int i = 1; i <= 10; i++) {
            System.out.print(i + " ");
        }
```

```
        System.out.println("\n\nEven numbers:");
        for (int i = 2; i <= 20; i += 2) {
            System.out.print(i + " ");
        }

        // Nested loop - multiplication table
        System.out.println("\n\nMultiplication Table (5x5):");
        for (int i = 1; i <= 5; i++) {
            for (int j = 1; j <= 5; j++) {
                System.out.printf("%4d", i * j);
            }
            System.out.println();
        }

        // Enhanced for loop (for-each)
        int[] numbers = {10, 20, 30, 40, 50};
        System.out.println("\nArray elements:");
        for (int num : numbers) {
            System.out.print(num + " ");
        }
    }
```

```
}
```

while and do-while Loops

```
public class WhileLoopDemo {
```

```
    public static void main(String[] args) {
        // while loop
        int count = 1;
        while (count <= 5) {
            System.out.println("Count: " + count);
            count++;
        }

        // do-while loop (executes at least once)
        int number = 10;
        do {
            System.out.println("Number is: " + number);
            number--;
        } while (number >= 5);
```

```
        // Infinite loop with break
        int i = 0;
        while (true) {
            System.out.print(i + " ");
            i++;
            if (i >= 10) break;
        }

        // continue statement
        System.out.println("\n\nOdd numbers:");
        for (int j = 1; j <= 10; j++) {
            if (j % 2 == 0) continue;  // skip even numbers
            System.out.print(j + " ");
        }
    }
}
```

---

5. Using Blocks of Code

A **block** is a group of statements enclosed in curly braces {}.

```
public class BlockDemo {

    public static void main(String[] args) {
        int x = 10;

        // Simple block
        {
            int y = 20;  // y exists only inside this block
            System.out.println("Inside block: x = " + x + ", y = " + y);
        }
        // System.out.println(y);  // ERROR - y not accessible here

        // Block in if statement
        if (x > 0) {
            System.out.println("x is positive");
```

```
        int square = x * x;
        System.out.println("Square: " + square);
    }

    // Block in loop
    for (int i = 1; i <= 3; i++) {
        int cube = i * i * i;
        System.out.println("Cube of " + i + " = " + cube);
    }

    // Nested blocks
    {
        int a = 5;
        {
            int b = 10;
            {
                int c = 15;
                System.out.println("Nested: " + a + ", " + b + ", " + c);
            }
            // c not accessible here
            System.out.println("Still have: " + a + ", " + b);
        }
        System.out.println("Still have: " + a);
    }
}
```

---

6. Lexical Issues (Tokens)

A Java program consists of **tokens** - the smallest individual elements recognized by the compiler.

Types of Tokens:

```
public class TokenDemo {

    public static void main(String[] args) {
```

```
    // 1. Keywords (reserved words)
    int number = 100;       // 'int' is keyword
    if (number > 50) {      // 'if' is keyword
        System.out.println("Number is large");
    }

    // 2. Identifiers (names given by programmer)
    int studentAge = 20;    // 'studentAge' is identifier
    String firstName = "John"; // 'firstName' is identifier
    double salaryAmount = 50000; // 'salaryAmount' is identifier

    // 3. Literals (constant values)
    int intLiteral = 42;        // integer literal
    double doubleLiteral = 3.14;  // floating-point literal
    char charLiteral = 'A';     // character literal
    String stringLiteral = "Hello"; // string literal
    boolean boolLiteral = true;   // boolean literal

    // 4. Operators
    int sum = 10 + 20;      // + is operator
    int product = 5 * 3;    // * is operator
    boolean isEqual = (10 == 10); // == is operator

    // 5. Separators
    // ( ) { } [ ] ; , . are separators
    int[] arr = {1, 2, 3};   // { } [ ] , are separators

    System.out.println("Sum: " + sum);  // ; and . are separators
  }
```

```
}
```

Java Keywords (53 reserved words)

Category	Keywords
Data Types	byte, short, int, long, float, double, char, boolean
Flow Control	if, else, switch, case, default, for, while, do, break, continue, return
Object Oriented	class, interface, extends, implements, new, this, super, instanceof

Category	Keywords
Access Modifiers	public, private, protected
Exception Handling	try, catch, finally, throw, throws
Class Related	static, final, abstract, synchronized, volatile, transient, native
Package Related	package, import
Other	void, enum, assert, strictfp
Unused	const, goto (reserved but not used)

Identifiers Rules:

```
public class IdentifierRules {
```

```
    public static void main(String[] args) {
        // Valid identifiers
        int age = 25;
        int _value = 10;          // Can start with underscore
        int $price = 100;         // Can start with dollar sign
        int studentAge1 = 20;     // Can contain numbers (not at start)
        int my_name = 30;         // Can use underscore
        int camelCaseVariable = 40; // Convention: camelCase

        // Invalid identifiers (uncomment to see errors)
        // int 1stNumber = 10;     // Cannot start with number
        // int my-name = 20;       // Cannot use hyphen
        // int class = 30;         // Cannot use keyword
        // int @value = 40;        // Cannot use special characters except _ and $

        // Naming conventions
        String studentName;       // variables: camelCase
        final int MAX_VALUE = 100;  // constants: UPPER_SNAKE_CASE
        class StudentRecord { }     // classes: PascalCase
        void calculateTotal() { }   // methods: camelCase
    }
```

```
}
```

---

7. Java Class Libraries

Java provides a rich set of built-in classes organized into **packages**.

Commonly Used Packages:

```
import java.util.*;     // Utility classes (Scanner, ArrayList, HashMap)
```

```
import java.io.*;       // Input/Output classes
import java.time.*;     // Date and Time API
import java.math.*;     // Big integers and decimals
import java.net.*;      // Networking
import java.sql.*;      // Database connectivity

public class ClassLibrariesDemo {
    public static void main(String[] args) {
        // java.lang - automatically imported (no need to import)
        String str = "Hello";          // String class
        System.out.println(str.length()); // System class
        Math.sqrt(25);                 // Math class

        // java.util
        Scanner sc = new Scanner(System.in);
        ArrayList<String> list = new ArrayList<>();
        HashMap<String, Integer> map = new HashMap<>();

        // java.time
        LocalDate today = LocalDate.now();
        LocalDateTime now = LocalDateTime.now();

        // java.math
        BigInteger bigNum = new BigInteger("12345678901234567890");
        BigDecimal precise = new BigDecimal("99.99");

        System.out.println("Today: " + today);

        sc.close();
    }
```

```
}
```

Commonly Used Classes:

Class	Package	Purpose
String	java.lang	String manipulation
StringBuilder	java.lang	Mutable strings
Math	java.lang	Mathematical functions
System	java.lang	System operations
Scanner	java.util	Input parsing
ArrayList	java.util	Dynamic array
HashMap	java.util	Key-value pairs
File	java.io	File operations
BufferedReader	java.io	Efficient reading

---

8. Java is a Strongly Typed Language

Java is a **strongly typed language**, meaning every variable must have a declared type, and the compiler strictly checks type compatibility. This helps catch errors early in the development process.

```
public class StronglyTypedDemo {

    public static void main(String[] args) {
        int number = 10;      // declared as int
        // number = "Hello";   // ERROR: cannot assign String to int

        double pi = 3.14;     // declared as double
        // pi = true;         // ERROR: cannot assign boolean to double

        String name = "Alice"; // declared as String
        // name = 123;         // ERROR: cannot assign int to String
```

```
    // Type compatibility is checked at compile time
    int result = 10 + 20;       // OK: int + int = int
    double result2 = 10 + 3.14;  // OK: int + double = double

    // Type conversion (casting) is explicit
    double d = 10.5;
    int i = (int) d;  // Explicit cast required
  }
```

```
}
```

---

9. Primitive Data Types

Java has **8 primitive data types**, which are the basic building blocks for data manipulation. They are stored directly in memory and are not objects.

```
public class PrimitiveTypesDemo {
```

```
  public static void main(String[] args) {
    // 1. byte (1 byte = 8 bits) -128 to 127
    byte b = 100;
    System.out.println("byte: " + b);

    // 2. short (2 bytes) -32,768 to 32,767
    short s = 30000;
    System.out.println("short: " + s);

    // 3. int (4 bytes) -2,147,483,648 to 2,147,483,647
    int i = 2000000;
    System.out.println("int: " + i);

    // 4. long (8 bytes) -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
    long l = 9000000000L;  // Note: 'L' suffix required
    System.out.println("long: " + l);

    // 5. float (4 bytes) ±3.4e-38 to ±3.4e+38
    float f = 3.14f;  // Note: 'f' suffix required
    System.out.println("float: " + f);
```

```
        // 6. double (8 bytes) ±1.7e-308 to ±1.7e+308
        double d = 3.141592653589793;
        System.out.println("double: " + d);

        // 7. char (2 bytes) 0 to 65,535 (Unicode)
        char c = 'A';
        char unicode = '\u0041';  // Unicode for 'A'
        System.out.println("char: " + c);
        System.out.println("unicode: " + unicode);

        // 8. boolean (1 bit) true/false
        boolean bool = true;
        System.out.println("boolean: " + bool);

        // Default values (for class members, not local variables)
        System.out.println("\nDefault values:");
        System.out.println("byte: 0, short: 0, int: 0, long: 0L");
        System.out.println("float: 0.0f, double: 0.0d");
        System.out.println("char: '\u0000', boolean: false");
    }
```

```
}
```

Data Type Ranges and Sizes

Type	Size	Range	Default	Example
byte	1 byte	-128 to 127	0	byte b = 100;
short	2 bytes	-32,768 to 32,767	0	short s = 1000;
int	4 bytes	-2^{31} to $2^{31}-1$	0	int i = 50000;
long	8 bytes	-2^{63} to $2^{63}-1$	0L	long l = 100000L;
float	4 bytes	±3.4e-38 to ±3.4e+38	0.0f	float f = 3.14f;
double	8 bytes	±1.7e-308 to ±1.7e+308	0.0d	double d = 3.14;
char	2 bytes	0 to 65,535	'\u0000'	char c = 'A';

Type	Size	Range	Default	Example
boolean	1 bit	true/false	false	boolean b = true;

Integer Literals

```
public class IntegerLiteralsDemo {
```

```
    public static void main(String[] args) {
        // Decimal (base 10)
        int decimal = 42;
        System.out.println("Decimal: " + decimal);

        // Binary (base 2) - prefix 0b or 0B
        int binary = 0b101010;  // 42 in decimal
        System.out.println("Binary 0b101010 = " + binary);

        // Octal (base 8) - prefix 0
        int octal = 052;       // 42 in decimal
        System.out.println("Octal 052 = " + octal);

        // Hexadecimal (base 16) - prefix 0x or 0X
        int hex = 0x2A;        // 42 in decimal
        System.out.println("Hex 0x2A = " + hex);

        // Underscores for readability (Java 7+)
        int million = 1_000_000;
        long creditCard = 1234_5678_9012_3456L;
        System.out.println("Million: " + million);

        // Type suffixes
        long bigNumber = 100L;         // L or l for long
        float pi = 3.14f;              // F or f for float
        double d = 3.14;               // D or d optional for double
    }
```

```
}
```

Floating-Point Literals

```
public class FloatLiteralsDemo {
```

```
    public static void main(String[] args) {
        // Standard notation
        double d1 = 3.14;
        double d2 = .5;     // 0.5
        double d3 = 42.0;   // 42.0

        // Scientific notation
        double scientific1 = 1.23e4;   // 1.23 × 10⁴ = 12300
        double scientific2 = 1.23E-4;  // 1.23 × 10⁻⁴ = 0.000123

        System.out.println("1.23e4 = " + scientific1);
        System.out.println("1.23e-4 = " + scientific2);

        // Special values
        double infinity = Double.POSITIVE_INFINITY;
        double negativeInfinity = Double.NEGATIVE_INFINITY;
        double nan = Double.NaN;  // Not a Number

        System.out.println("Infinity: " + (10.0 / 0));
        System.out.println("NaN: " + (0.0 / 0));
    }
```

```
}
```

Character Literals

```
public class CharLiteralsDemo {
```

```
    public static void main(String[] args) {
        // Single quote character
        char c1 = 'A';
        char c2 = '9';
        char c3 = '$';

        // Escape sequences
        char newline = '\n';
        char tab = '\t';
        char backspace = '\b';
        char carriageReturn = '\r';
        char formFeed = '\f';
        char backslash = '\\';
        char singleQuote = '\'';
```

```
        char doubleQuote = '\"';

        // Unicode representation
        char omega = '\u03A9';  // Greek Omega
        char smiley = '\u263A';  // Smiley face

        System.out.println("Omega: " + omega);
        System.out.println("Smiley: " + smiley);

        // char can be used as integer (0-65535)
        int code = 'A';
        System.out.println("ASCII of A: " + code);

        // Increment char
        char letter = 'A';
        letter++;
        System.out.println("Next letter: " + letter);  // B
    }
```

```
}
```

Boolean Literals

```
public class BooleanDemo {
```

```
    public static void main(String[] args) {
        boolean isJavaFun = true;
        boolean isFishTasty = false;

        System.out.println("Is Java fun? " + isJavaFun);
        System.out.println("Is fish tasty? " + isFishTasty);

        // Boolean expressions
        int a = 10, b = 20;
        boolean isGreater = a > b;
        boolean isEqual = (a == b);
        boolean result = (a < b) && (b > 0);

        System.out.println("10 > 20: " + isGreater);
        System.out.println("10 == 20: " + isEqual);
        System.out.println("Result: " + result);
```

```
    // Cannot use 0/1 like C/C++
    // boolean wrong = 1;  // ERROR
  }
```

```
}
```

---

10. Variables

Variable Declaration and Initialization

```
public class VariableDemo {
```

```
  public static void main(String[] args) {
    // Declaration
    int age;
    String name;
    double salary;

    // Initialization
    age = 25;
    name = "John Doe";
    salary = 50000.50;

    // Declaration and initialization together
    int count = 10;
    String message = "Hello";
    double pi = 3.14159;

    // Multiple declarations
    int a, b, c;
    int x = 1, y = 2, z = 3;

    // Variables can be reassigned
    age = 26;  // changes from 25 to 26

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

```
}
```

Variable Scope

```
public class ScopeDemo {
    // Class variable (instance variable)
    int instanceVar = 10;

    // Static variable (class variable)
    static int staticVar = 20;

    public void method1() {
        // Local variable
        int localVar = 30;
        System.out.println("Instance: " + instanceVar);
        System.out.println("Static: " + staticVar);
        System.out.println("Local: " + localVar);
    }

    public void method2() {
        // Can access instance and static variables
        System.out.println("Instance: " + instanceVar);
        System.out.println("Static: " + staticVar);
        // System.out.println(localVar);  // ERROR - localVar not accessible

        // Block scope
        if (true) {
            int blockVar = 40;
            System.out.println("Block variable: " + blockVar);
        }
        // System.out.println(blockVar);  // ERROR - blockVar out of scope

        // Loop scope
        for (int i = 0; i < 5; i++) {
            System.out.println("i = " + i);
        }
        // System.out.println(i);  // ERROR - i out of scope
    }

    public static void main(String[] args) {
        ScopeDemo obj = new ScopeDemo();
```

```
        obj.method1();
        obj.method2();

        // Local variable in main
        int mainVar = 50;
        System.out.println("Main variable: " + mainVar);
    }
```

```
}
```

Type Conversion and Casting

Implicit (Widening) Conversion - Automatic

```
public class ImplicitConversionDemo {
```

```
    public static void main(String[] args) {
        // byte → short → int → long → float → double

        byte b = 100;
        short s = b;       // byte to short (automatic)
        int i = s;         // short to int (automatic)
        long l = i;        // int to long (automatic)
        float f = l;       // long to float (automatic - possible precision loss)
        double d = f;      // float to double (automatic)

        System.out.println("byte: " + b);
        System.out.println("short: " + s);
        System.out.println("int: " + i);
        System.out.println("long: " + l);
        System.out.println("float: " + f);
        System.out.println("double: " + d);

        // char to int
        char ch = 'A';
        int code = ch;  // ASCII value 65
        System.out.println("Char 'A' to int: " + code);
    }
```

```
}
```

Explicit (Narrowing) Conversion - Casting Required

```
public class ExplicitConversionDemo {
```

```
    public static void main(String[] args) {
        // double → float → long → int → short → byte

        double d = 100.99;
        float f = (float) d;     // double to float
        long l = (long) f;       // float to long (truncates decimal)
        int i = (int) l;         // long to int
        short s = (short) i;     // int to short (may lose data)
        byte b = (byte) s;       // short to byte (may lose data)

        System.out.println("double: " + d);
        System.out.println("float: " + f);
        System.out.println("long: " + l);
        System.out.println("int: " + i);
        System.out.println("short: " + s);
        System.out.println("byte: " + b);

        // Casting overflow example
        int large = 300;
        byte small = (byte) large;  // 300 % 256 = 44
        System.out.println("300 cast to byte: " + small);

        // Casting char to int and back
        char ch = 'Z';
        int num = ch;
        char back = (char) num;
        System.out.println("Char: " + ch + ", Int: " + num + ", Back: " + back);
    }
```

```
}
```

Automatic Type Promotion in Expressions

```
public class TypePromotionDemo {
```

```
    public static void main(String[] args) {
        // Rule 1: byte, short, char are promoted to int
        byte b = 10;
        short s = 20;
```

```
char c = 30;
int result1 = b + s + c;  // All promoted to int
System.out.println("byte+short+char = " + result1);

// Rule 2: If one operand is long, entire expression promoted to long
int i = 10;
long l = 20L;
long result2 = i + l;  // int promoted to long
System.out.println("int+long = " + result2);

// Rule 3: If one operand is float, entire expression promoted to float
float f = 10.5f;
int x = 5;
float result3 = f + x;  // int promoted to float
System.out.println("float+int = " + result3);

// Rule 4: If one operand is double, entire expression promoted to double
double d = 10.5;
int y = 5;
double result4 = d + y;  // int promoted to double
System.out.println("double+int = " + result4);

// Complete example
byte b2 = 42;
char c2 = 'a';
short s2 = 1024;
int i2 = 50000;
float f2 = 5.67f;
double d2 = .1234;

double result = (f2 * b2) + (i2 / c2) - (d2 * s2);
// f2*b2 → float
// i2/c2 → int (c2 promoted to int)
// d2*s2 → double (s2 promoted to double)
// float + int → float
// float - double → double
System.out.println("Complex expression: " + result);

// Be careful with overflow
byte b3 = 50;
byte b4 = 100;
// byte sum = b3 + b4;  // ERROR: needs explicit cast
byte sum = (byte)(b3 + b4);  // 150 - works
System.out.println("Sum: " + sum);
```

```
    }
}
```

---

11. Arrays

Arrays are objects that store multiple values of the same type in a contiguous memory location. They are fixed in size once created.

One-Dimensional Arrays

```
public class OneDArrayDemo {

    public static void main(String[] args) {
        // Method 1: Declaration and allocation
        int[] numbers = new int[5];

        // Method 2: Declaration, allocation, initialization
        int[] scores = {85, 90, 75, 88, 92};

        // Method 3: Using new with initializer list
        int[] values = new int[]{10, 20, 30, 40, 50};

        // Setting values
        numbers[0] = 10;
        numbers[1] = 20;
        numbers[2] = 30;
        numbers[3] = 40;
        numbers[4] = 50;

        // Accessing values
        System.out.println("First element: " + numbers[0]);
        System.out.println("Array length: " + numbers.length);

        // Traversing array
        System.out.println("\nAll elements:");
        for (int i = 0; i < scores.length; i++) {
            System.out.println("scores[" + i + "] = " + scores[i]);
```

```
        }

        // Enhanced for loop (for-each)
        System.out.println("\nUsing for-each:");
        for (int score : scores) {
            System.out.print(score + " ");
        }

        // Finding sum and average
        int sum = 0;
        for (int num : numbers) {
            sum += num;
        }
        double average = (double) sum / numbers.length;
        System.out.println("\n\nSum: " + sum);
        System.out.println("Average: " + average);

        // Finding min and max
        int min = scores[0];
        int max = scores[0];
        for (int score : scores) {
            if (score < min) min = score;
            if (score > max) max = score;
        }
        System.out.println("Min: " + min + ", Max: " + max);
    }
}
```

Multidimensional Arrays

```
public class MultiDimensionalArrayDemo {

    public static void main(String[] args) {
        // 2D array - matrix
        int[][] matrix = {
            {1, 2, 3},
            {4, 5, 6},
            {7, 8, 9}
        };

        System.out.println("Matrix:");
```

```
for (int i = 0; i < matrix.length; i++) {
    for (int j = 0; j < matrix[i].length; j++) {
        System.out.print(matrix[i][j] + " ");
    }
    System.out.println();
}

// 2D array using new
int[][] table = new int[3][4];

// Filling the table
for (int i = 0; i < table.length; i++) {
    for (int j = 0; j < table[i].length; j++) {
        table[i][j] = i * j;
    }
}

// Jagged array (rows with different lengths)
int[][] jagged = new int[3][];
jagged[0] = new int[]{1};
jagged[1] = new int[]{2, 3};
jagged[2] = new int[]{4, 5, 6};

System.out.println("\nJagged array:");
for (int i = 0; i < jagged.length; i++) {
    for (int val : jagged[i]) {
        System.out.print(val + " ");
    }
    System.out.println();
}

// 3D array
int[][][] cube = new int[2][3][4];
System.out.println("\n3D array dimensions: " + cube.length + " x " +
                   cube[0].length + " x " + cube[0][0].length);

// Array of objects
String[] names = new String[3];
names[0] = "Alice";
names[1] = "Bob";
names[2] = "Charlie";

System.out.println("\nNames:");
for (String name : names) {
```

```
        System.out.println(name);
    }
  }
```

```
}
```

Array Operations

```
import java.util.Arrays;
```

```


public class ArrayOperationsDemo {
  public static void main(String[] args) {
    int[] numbers = {5, 2, 8, 1, 9, 3, 7, 4, 6};

    // Sorting
    Arrays.sort(numbers);
    System.out.println("Sorted: " + Arrays.toString(numbers));

    // Binary search (array must be sorted)
    int index = Arrays.binarySearch(numbers, 7);
    System.out.println("Index of 7: " + index);

    // Copy array
    int[] copy = Arrays.copyOf(numbers, numbers.length);
    int[] partial = Arrays.copyOfRange(numbers, 2, 6);

    System.out.println("Full copy: " + Arrays.toString(copy));
    System.out.println("Partial copy (2-5): " + Arrays.toString(partial));

    // Fill array with value
    int[] filled = new int[5];
    Arrays.fill(filled, 10);
    System.out.println("Filled array: " + Arrays.toString(filled));

    // Compare arrays
    int[] arr1 = {1, 2, 3};
    int[] arr2 = {1, 2, 3};
    int[] arr3 = {1, 2, 4};

    System.out.println("arr1 equals arr2: " + Arrays.equals(arr1, arr2));
    System.out.println("arr1 equals arr3: " + Arrays.equals(arr1, arr3));
```

```
        // Convert to list
        String[] words = {"apple", "banana", "orange"};
        java.util.List<String> list = Arrays.asList(words);
        System.out.println("List: " + list);

        // 2D array operations
        int[][] m1 = {{1, 2}, {3, 4}};
        int[][] m2 = {{1, 2}, {3, 4}};

        System.out.println("2D arrays equal: " + Arrays.deepEquals(m1, m2));
        System.out.println("2D array string: " + Arrays.deepToString(m1));
    }
}
```

---

12. A Few Words About Strings

Strings in Java are objects, not primitive types. They are immutable, meaning their value cannot be changed after creation. Any operation that appears to modify a string actually creates a new string object.

```
public class StringDemo {

    public static void main(String[] args) {
        // Creating strings
        String str1 = "Hello";          // String literal (pooled)
        String str2 = new String("World");  // Using new keyword
        String str3 = "Hello";          // Same as str1 (same object)

        // String comparison
        System.out.println("str1 == str3: " + (str1 == str3));      // true (same object)
        System.out.println("str1 == str2: " + (str1 == str2));      // false (different objects)
        System.out.println("str1 equals str2: " + str1.equals(str2)); // false
        System.out.println("str1 equals 'Hello': " + str1.equals("Hello")); // true

        // String methods
        String text = "Java Programming";
```

```
        System.out.println("Length: " + text.length());
        System.out.println("Char at 0: " + text.charAt(0));
        System.out.println("Substring (0-4): " + text.substring(0, 4));
        System.out.println("Lowercase: " + text.toLowerCase());
        System.out.println("Uppercase: " + text.toUpperCase());
        System.out.println("Contains 'Program': " + text.contains("Program"));
        System.out.println("Index of 'a': " + text.indexOf('a'));
        System.out.println("Last index of 'a': " + text.lastIndexOf('a'));
        System.out.println("Replace 'a' with 'x': " + text.replace('a', 'x'));

        // String concatenation
        String s1 = "Hello";
        String s2 = "World";
        String s3 = s1 + " " + s2;
        System.out.println("Concatenated: " + s3);

        // Using concat() method
        String s4 = s1.concat(" ").concat(s2);
        System.out.println("Using concat(): " + s4);

        // String formatting
        String formatted = String.format("Name: %s, Age: %d", "Alice", 25);
        System.out.println(formatted);

        // Splitting strings
        String csv = "apple,banana,orange,grape";
        String[] fruits = csv.split(",");
        System.out.println("Split result:");
        for (String fruit : fruits) {
            System.out.println("  " + fruit);
        }

        // Trimming whitespace
        String withSpaces = "  Hello  ";
        System.out.println("Trimmed: '" + withSpaces.trim() + "'");

        // Checking empty/blank
        String empty = "";
        String blank = "   ";
        System.out.println("isEmpty: " + empty.isEmpty());
        System.out.println("isBlank: " + blank.isBlank());  // Java 11+
    }
```

```
}
```

String Immutability

```
public class StringImmutabilityDemo {
    public static void main(String[] args) {
        String str = "Hello";
        System.out.println("Original: " + str);
        System.out.println("HashCode: " + System.identityHashCode(str));

        // This creates a NEW string, doesn't modify the original
        str = str.concat(" World");
        System.out.println("After concat: " + str);
        System.out.println("HashCode: " + System.identityHashCode(str));  // Different!

        // StringBuilder for mutable strings
        StringBuilder sb = new StringBuilder("Hello");
        System.out.println("\nStringBuilder original: " + sb);
        System.out.println("HashCode: " + System.identityHashCode(sb));

        sb.append(" World");
        System.out.println("After append: " + sb);
        System.out.println("HashCode: " + System.identityHashCode(sb));  // Same!

        // Performance comparison
        long start = System.currentTimeMillis();
        String s = "";
        for (int i = 0; i < 10000; i++) {
            s += i;  // Creates many intermediate objects
        }
        long end = System.currentTimeMillis();
        System.out.println("\nString concatenation time: " + (end - start) + "ms");

        start = System.currentTimeMillis();
        StringBuilder sb2 = new StringBuilder();
        for (int i = 0; i < 10000; i++) {
            sb2.append(i);
        }
        end = System.currentTimeMillis();
        System.out.println("StringBuilder time: " + (end - start) + "ms");
    }
}
```

13. Arithmetic Operators

Arithmetic operators are used to perform mathematical calculations.

```
public class ArithmeticOperatorsDemo {
```

```
    public static void main(String[] args) {
        int a = 10, b = 3;

        // Basic arithmetic
        System.out.println("a + b = " + (a + b));  // 13
        System.out.println("a - b = " + (a - b));  // 7
        System.out.println("a * b = " + (a * b));  // 30
        System.out.println("a / b = " + (a / b));  // 3 (integer division)
        System.out.println("a % b = " + (a % b));  // 1 (modulus/remainder)

        // Integer division truncates
        System.out.println("\nInteger division:");
        System.out.println("10 / 3 = " + (10 / 3));     // 3
        System.out.println("10 / 3.0 = " + (10 / 3.0));  // 3.333...

        // Modulus with negative numbers
        System.out.println("\nModulus with negatives:");
        System.out.println("10 % 3 = " + (10 % 3));   // 1
        System.out.println("-10 % 3 = " + (-10 % 3)); // -1
        System.out.println("10 % -3 = " + (10 % -3)); // 1

        // Increment and decrement operators
        int x = 5;
        System.out.println("\nIncrement/Decrement:");
        System.out.println("x = " + x);
        System.out.println("++x = " + (++x));  // Pre-increment: 6
        System.out.println("x++ = " + (x++));  // Post-increment: 6 (then x becomes 7)
        System.out.println("x = " + x);       // 7
        System.out.println("--x = " + (--x));  // Pre-decrement: 6
        System.out.println("x-- = " + (x--));  // Post-decrement: 6 (then x becomes 5)
        System.out.println("x = " + x);       // 5

        // Compound assignments
        int y = 10;
```

```
        y += 5;    // y = y + 5
        System.out.println("\nCompound assignments:");
        System.out.println("y += 5: " + y);
        y -= 3;    // y = y - 3
        System.out.println("y -= 3: " + y);
        y *= 2;    // y = y * 2
        System.out.println("y *= 2: " + y);
        y /= 4;    // y = y / 4
        System.out.println("y /= 4: " + y);
        y %= 3;    // y = y % 3
        System.out.println("y %= 3: " + y);
    }
```

```
}
```

---

14. Bitwise Operators

Bitwise operators work on individual bits of integer types. They are used for low-level programming, such as manipulating flags or optimizing certain calculations.

```
public class BitwiseOperatorsDemo {
```

```
    public static void main(String[] args) {
        int a = 5;  // Binary: 0101
        int b = 3;  // Binary: 0011

        // Bitwise AND (&) - 1 if both bits are 1
        System.out.println("a & b = " + (a & b));  // 0101 & 0011 = 0001 (1)

        // Bitwise OR (|) - 1 if at least one bit is 1
        System.out.println("a | b = " + (a | b));  // 0101 | 0011 = 0111 (7)

        // Bitwise XOR (^) - 1 if bits are different
        System.out.println("a ^ b = " + (a ^ b));  // 0101 ^ 0011 = 0110 (6)

        // Bitwise NOT (~) - flips all bits (one's complement)
        System.out.println("~a = " + (~a));       // ~0101 = 1010... (depends on 32-bit)

        // Left shift (<<) - shifts bits left, fills with 0
```

```
        System.out.println("a << 1 = " + (a << 1));  // 0101 << 1 = 1010 (10)
        System.out.println("a << 2 = " + (a << 2));  // 0101 << 2 = 10100 (20)

        // Right shift (>>) - shifts bits right, preserves sign
        System.out.println("a >> 1 = " + (a >> 1));  // 0101 >> 1 = 0010 (2)

        // Unsigned right shift (>>>) - shifts bits right, fills with 0
        int negative = -5;
        System.out.println("negative = " + negative);
        System.out.println("negative >> 1 = " + (negative >> 1));   // Preserves sign
        System.out.println("negative >>> 1 = " + (negative >>> 1)); // Fills with 0

        // Practical use: Checking if number is even/odd
        int num = 7;
        if ((num & 1) == 0) {
            System.out.println(num + " is even");
        } else {
            System.out.println(num + " is odd");
        }

        // Practical use: Power of 2 check
        int power = 16;
        if ((power & (power - 1)) == 0) {
            System.out.println(power + " is power of 2");
        }

        // Practical use: Toggle a bit
        int flags = 0b0000;
        flags = flags ^ (1 << 2);  // Set bit 2
        System.out.println("After toggling bit 2: " + Integer.toBinaryString(flags));
        flags = flags ^ (1 << 2);  // Toggle back
        System.out.println("After toggling again: " + Integer.toBinaryString(flags));
    }
```

```
}
```

---

15. Relational Operators

Relational operators are used to compare two values and return a boolean result (true or false).

```
public class RelationalOperatorsDemo {
    public static void main(String[] args) {
        int a = 10, b = 20, c = 10;

        // Equality
        System.out.println("a == b: " + (a == b));  // false
        System.out.println("a == c: " + (a == c));  // true

        // Inequality
        System.out.println("a != b: " + (a != b));  // true
        System.out.println("a != c: " + (a != c));  // false

        // Greater than
        System.out.println("a > b: " + (a > b));   // false
        System.out.println("b > a: " + (b > a));   // true

        // Less than
        System.out.println("a < b: " + (a < b));   // true
        System.out.println("b < a: " + (b < a));   // false

        // Greater than or equal
        System.out.println("a >= c: " + (a >= c));  // true
        System.out.println("a >= b: " + (a >= b));  // false

        // Less than or equal
        System.out.println("a <= c: " + (a <= c));  // true
        System.out.println("b <= a: " + (b <= a));  // false

        // Comparing objects (reference comparison)
        String str1 = "Hello";
        String str2 = "Hello";
        String str3 = new String("Hello");

        System.out.println("\nString reference comparison:");
        System.out.println("str1 == str2: " + (str1 == str2));         // true
        System.out.println("str1 == str3: " + (str1 == str3));         // false
        System.out.println("str1.equals(str3): " + str1.equals(str3));   // true
    }
}
```

16. Boolean Logical Operators

Boolean logical operators combine boolean expressions and return a boolean result.

```
public class LogicalOperatorsDemo {
```

```
    public static void main(String[] args) {
        boolean p = true;
        boolean q = false;

        // Logical AND (&&) - short-circuit
        System.out.println("p && q: " + (p && q));  // false
        System.out.println("p && true: " + (p && true));  // true

        // Logical OR (||) - short-circuit
        System.out.println("p || q: " + (p || q));  // true
        System.out.println("q || false: " + (q || false));  // false

        // Logical NOT (!)
        System.out.println("!p: " + (!p));  // false
        System.out.println("!q: " + (!q));  // true

        // Bitwise logical operators (no short-circuit)
        System.out.println("\nBitwise logical (no short-circuit):");
        System.out.println("p & q: " + (p & q));   // false
        System.out.println("p | q: " + (p | q));   // true
        System.out.println("p ^ q: " + (p ^ q));   // true (XOR)

        // Short-circuit demonstration
        System.out.println("\nShort-circuit behavior:");
        int x = 10;

        // Second condition not evaluated because first is false
        if (x > 20 && (x++ > 0)) {
            // This won't execute
        }
        System.out.println("x after short-circuit AND: " + x);  // Still 10

        // Both conditions evaluated
        if (x > 20 & (x++ > 0)) {
```

```
        // This won't execute
    }
    System.out.println("x after bitwise AND: " + x);  // 11

    // Truth table
    System.out.println("\nTruth Table:");
    System.out.println("A\tB\tA&&B\tA||B\tA^B\t!A");
    System.out.println("true\ttrue\t" + (true&&true) + "\t" + (true||true) +
                       "\t" + (true^true) + "\t" + (!true));
    System.out.println("true\tfalse\t" + (true&&false) + "\t" + (true||false) +
                       "\t" + (true^false) + "\t" + (!true));
    System.out.println("false\ttrue\t" + (false&&true) + "\t" + (false||true) +
                       "\t" + (false^true) + "\t" + (!false));
    System.out.println("false\tfalse\t" + (false&&false) + "\t" + (false||false) +
                       "\t" + (false^false) + "\t" + (!false));
  }
}
```

---

17. The Assignment Operator

The assignment operator (=) is used to assign a value to a variable. Compound assignment operators (+=, -=, *=, etc.) combine an arithmetic or bitwise operation with an assignment.

```
public class AssignmentOperatorsDemo {

  public static void main(String[] args) {
    // Simple assignment
    int a = 10;
    System.out.println("a = " + a);

    // Multiple assignments (right-associative)
    int b, c;
    b = c = 20;  // Equivalent to: c = 20; b = c;
    System.out.println("b = " + b + ", c = " + c);

    // Compound assignments
    int x = 5;
```

```
        x += 3;   // x = x + 3
        System.out.println("x += 3: " + x);

        x -= 2;   // x = x - 2
        System.out.println("x -= 2: " + x);

        x *= 4;   // x = x * 4
        System.out.println("x *= 4: " + x);

        x /= 3;   // x = x / 3
        System.out.println("x /= 3: " + x);

        x %= 2;   // x = x % 2
        System.out.println("x %= 2: " + x);

        // Bitwise compound assignments
        int y = 5;  // 0101
        y &= 3;    // 0101 & 0011 = 0001
        System.out.println("y &= 3: " + y);

        y |= 2;    // 0001 | 0010 = 0011
        System.out.println("y |= 2: " + y);

        y ^= 1;    // 0011 ^ 0001 = 0010
        System.out.println("y ^= 1: " + y);

        y <<= 1;   // 0010 << 1 = 0100
        System.out.println("y <<= 1: " + y);

        y >>= 1;   // 0100 >> 1 = 0010
        System.out.println("y >>= 1: " + y);

        // Assignment with object references
        String s1 = "Hello";
        String s2 = s1;  // s2 references same object as s1
        System.out.println("\ns1 = " + s1);
        System.out.println("s2 = " + s2);
        System.out.println("s1 == s2: " + (s1 == s2));
    }
```

```
}
```

18. The ? Operator (Ternary Operator)

The ternary operator (? :) is a shorthand for a simple if-else statement. It takes three operands: a boolean condition, a value to return if the condition is true, and a value to return if the condition is false.

```
public class TernaryOperatorDemo {
```

```
    public static void main(String[] args) {
        // Basic ternary: condition ? value_if_true : value_if_false

        // Finding maximum
        int a = 10, b = 20;
        int max = (a > b) ? a : b;
        System.out.println("Max of " + a + " and " + b + " is " + max);

        // Checking even/odd
        int num = 7;
        String result = (num % 2 == 0) ? "Even" : "Odd";
        System.out.println(num + " is " + result);

        // Nested ternary
        int score = 85;
        char grade = (score >= 90) ? 'A' :
                     (score >= 80) ? 'B' :
                     (score >= 70) ? 'C' :
                     (score >= 60) ? 'D' : 'F';
        System.out.println("Score " + score + " = Grade " + grade);

        // Ternary with function calls
        int x = 5, y = 10;
        System.out.println("Larger is: " + ((x > y) ? x : y));

        // Ternary for conditional assignment
        int value = 100;
        int result2 = (value > 0) ? value : -value;  // absolute value
        System.out.println("Absolute value: " + result2);

        // Ternary with null check
        String name = null;
        String displayName = (name != null) ? name : "Guest";
        System.out.println("Display name: " + displayName);
```

```
        // Ternary vs if-else
        int age = 18;

        // Using if-else
        String status;
        if (age >= 18) {
            status = "Adult";
        } else {
            status = "Minor";
        }
        System.out.println("Using if-else: " + status);

        // Using ternary (more concise)
        status = (age >= 18) ? "Adult" : "Minor";
        System.out.println("Using ternary: " + status);
    }
}
```

---

19. Operator Precedence

Operator precedence determines the order in which operators are evaluated in an expression. Operators with higher precedence are evaluated before operators with lower precedence. When operators have the same precedence, their associativity (left-to-right or right-to-left) determines the order of evaluation.

```
public class OperatorPrecedenceDemo {

    public static void main(String[] args) {
        // Precedence (highest to lowest):
        // postfix: expr++ expr--
        // unary: ++expr --expr +expr -expr ~ !
        // multiplicative: * / %
        // additive: + -
        // shift: << >> >>>
        // relational: < > <= >= instanceof
        // equality: == !=
        // logical AND: &
```

```
// logical XOR: ^
// logical OR: |
// conditional AND: &&
// conditional OR: ||
// ternary: ? :
// assignment: = += -= *= etc.

// Example 1: Multiplication before addition
int result1 = 5 + 3 * 2;     // 5 + (3*2) = 11
System.out.println("5 + 3 * 2 = " + result1);

// Example 2: Parentheses change precedence
int result2 = (5 + 3) * 2;   // (5+3)*2 = 16
System.out.println("(5 + 3) * 2 = " + result2);

// Example 3: Unary operators
int a = 10;
int result3 = ++a * 2;       // (++a)*2 = 22
System.out.println("++a * 2 = " + result3);

// Example 4: Relational before logical
boolean result4 = 5 > 3 && 2 < 4;  // (5>3) && (2<4) = true
System.out.println("5 > 3 && 2 < 4 = " + result4);

// Example 5: Complex expression
int x = 10, y = 5, z = 2;
int result5 = x + y * z - (x / y) + z++;
// Step by step:
// y * z = 10
// x / y = 2
// z++ = 2 (then z becomes 3)
// x + 10 - 2 + 2 = 20
System.out.println("x + y * z - (x / y) + z++ = " + result5);

// Example 6: Using parentheses for clarity
int p = 2, q = 3, r = 4;
int result6 = (p + q) * r;         // (2+3)*4 = 20
int result7 = p + (q * r);         // 2 + (3*4) = 14
System.out.println("(p+q)*r = " + result6);
System.out.println("p+(q*r) = " + result7);

// Associativity (left to right for most operators)
int result8 = 10 - 5 - 2;          // (10-5)-2 = 3
System.out.println("10 - 5 - 2 = " + result8);
```

```
        // Assignment is right-associative
        int i, j, k;
        i = j = k = 5;                  // k=5; j=k; i=j;
        System.out.println("i = " + i + ", j = " + j + ", k = " + k);

        // instanceof operator
        String text = "Hello";
        boolean isString = text instanceof String;
        System.out.println("text instanceof String: " + isString);
    }
```

```
}
```

Operator Precedence Table

Precedence	Operators	Associativity
1 (highest)	++ -- (postfix)	Left to right
2	++ -- (prefix) + - (unary) ~ !	Right to left
3	* / %	Left to right
4	+ -	Left to right
5	<< >> >>>	Left to right
6	< > <= >= instanceof	Left to right
7	== !=	Left to right
8	&	Left to right
9	^	Left to right
10	`	`
11	&&	Left to right
12	`	
13	?:	Right to left

Precedence	Operators	Associativity
14 (lowest)	= += -= *= /= %= etc.	Right to left

---

20. Using Parentheses

Parentheses () are used to explicitly control the order of evaluation in expressions, overriding default operator precedence. They also improve readability.

```
public class ParenthesesDemo {

    public static void main(String[] args) {
        // Without parentheses (default precedence)
        int result1 = 10 + 20 / 5 - 2 * 3;
        // 10 + (20/5) - (2*3) = 10 + 4 - 6 = 8
        System.out.println("10 + 20 / 5 - 2 * 3 = " + result1);

        // With parentheses - changes order
        int result2 = (10 + 20) / (5 - 2) * 3;
        // (30) / (3) * 3 = 10 * 3 = 30
        System.out.println("(10 + 20) / (5 - 2) * 3 = " + result2);

        // Complex expressions with parentheses
        int a = 5, b = 3, c = 2, d = 4;

        int result3 = (a + b) * (c - d) / (a % b);
        System.out.println("(a+b)*(c-d)/(a%b) = " + result3);

        // Nested parentheses
        int result4 = ((a + b) * (c - d)) / ((a % b) + (b % c));
        System.out.println("Nested parentheses result: " + result4);

        // Boolean expressions
        boolean result5 = (a > b) && (c < d) || (a == c);
        // ((a>b) && (c<d)) || (a==c)
        System.out.println("(a>b) && (c<d) || (a==c) = " + result5);

        boolean result6 = (a > b) && ((c < d) || (a == c));
        System.out.println("(a>b) && ((c<d) || (a==c)) = " + result6);
```

```
        // Clarity - use parentheses even when not required
        int x = 10, y = 20, z = 30;
        int average = (x + y + z) / 3;  // Clear intent
        System.out.println("Average: " + average);

        // Bitwise with parentheses
        int flags = (1 << 2) | (1 << 5) | (1 << 7);
        System.out.println("Flags: " + Integer.toBinaryString(flags));
    }
```

```
}
```

---

21. Java's Selection Statements

Selection statements (or conditional statements) allow a program to make decisions and execute different blocks of code based on certain conditions.

if-else Statement

```
public class IfElseDemo {
```

```
    public static void main(String[] args) {
        int marks = 85;

        // Simple if
        if (marks >= 40) {
            System.out.println("You passed!");
        }

        // if-else
        if (marks >= 50) {
            System.out.println("You scored above 50");
        } else {
            System.out.println("You scored below 50");
        }

        // if-else-if ladder
```

```
        if (marks >= 90) {
            System.out.println("Grade: A+");
        } else if (marks >= 80) {
            System.out.println("Grade: A");
        } else if (marks >= 70) {
            System.out.println("Grade: B");
        } else if (marks >= 60) {
            System.out.println("Grade: C");
        } else if (marks >= 50) {
            System.out.println("Grade: D");
        } else {
            System.out.println("Grade: F");
        }

        // Nested if
        int age = 22;
        boolean hasLicense = true;

        if (age >= 18) {
            if (hasLicense) {
                System.out.println("You can drive");
            } else {
                System.out.println("Get a license first");
            }
        } else {
            System.out.println("Too young to drive");
        }

        // Multiple conditions
        int year = 2024;
        if ((year % 4 == 0 && year % 100 != 0) || (year % 400 == 0)) {
            System.out.println(year + " is a leap year");
        } else {
            System.out.println(year + " is not a leap year");
        }
    }
```

```
}
```

switch Statement

```
public class SwitchDemo {
```

```
public static void main(String[] args) {
    // Traditional switch
    int day = 3;
    String dayName;

    switch (day) {
        case 1:
            dayName = "Monday";
            break;
        case 2:
            dayName = "Tuesday";
            break;
        case 3:
            dayName = "Wednesday";
            break;
        case 4:
            dayName = "Thursday";
            break;
        case 5:
            dayName = "Friday";
            break;
        case 6:
            dayName = "Saturday";
            break;
        case 7:
            dayName = "Sunday";
            break;
        default:
            dayName = "Invalid day";
    }
    System.out.println("Day: " + dayName);

    // Switch with multiple cases
    char grade = 'B';
    String description;

    switch (grade) {
        case 'A':
        case 'A+':
            description = "Excellent";
            break;
        case 'B':
        case 'B+':
```

```
        description = "Good";
        break;
    case 'C':
        description = "Average";
        break;
    case 'D':
        description = "Poor";
        break;
    default:
        description = "Fail";
}
System.out.println("Grade " + grade + ": " + description);

// Switch with strings (Java 7+)
String fruit = "Apple";

switch (fruit.toLowerCase()) {
    case "apple":
        System.out.println("Apple is red or green");
        break;
    case "banana":
        System.out.println("Banana is yellow");
        break;
    case "orange":
        System.out.println("Orange is orange");
        break;
    default:
        System.out.println("Unknown fruit");
}

// Switch expressions (Java 14+)
int month = 2;
int days = switch (month) {
    case 1, 3, 5, 7, 8, 10, 12 -> 31;
    case 4, 6, 9, 11 -> 30;
    case 2 -> 28;
    default -> 0;
};
System.out.println("Month " + month + " has " + days + " days");

// Switch with yield (Java 13+)
int number = 2;
String result = switch (number) {
```

```
            case 1:
                yield "One";
            case 2:
                yield "Two";
            default:
                yield "Other";
        };
        System.out.println("Number " + number + " is " + result);

        // Fall-through demonstration (without break)
        int x = 2;
        System.out.print("Fall-through: ");
        switch (x) {
            case 1:
                System.out.print("One ");
            case 2:
                System.out.print("Two ");
            case 3:
                System.out.print("Three ");
            default:
                System.out.print("Default");
        }
        System.out.println(); // Output: Two Three Default
    }
}
```

---

22. Iteration Statements

Iteration statements (or loops) allow a block of code to be executed repeatedly until a certain condition is met.

for Loop

```
public class ForLoopDemo {

    public static void main(String[] args) {
        // Basic for loop
```

```
System.out.println("Numbers 1 to 10:");
for (int i = 1; i <= 10; i++) {
    System.out.print(i + " ");
}

// Count down
System.out.println("\n\nCountdown from 10:");
for (int i = 10; i >= 1; i--) {
    System.out.print(i + " ");
}

// Multiple variables
System.out.println("\n\nMultiple variables:");
for (int i = 0, j = 10; i <= 5; i++, j--) {
    System.out.println("i = " + i + ", j = " + j);
}

// Infinite loop (with break)
System.out.println("\nInfinite loop with break:");
int count = 0;
for (;;) {
    System.out.print(count + " ");
    count++;
    if (count >= 5) break;
}

// Empty loop (for delay)
System.out.println("\n\nEmpty loop (delay):");
long start = System.currentTimeMillis();
for (long i = 0; i < 100000000; i++);  // Empty loop body
long end = System.currentTimeMillis();
System.out.println("Time taken: " + (end - start) + "ms");

// Nested loops - multiplication table
System.out.println("\nMultiplication Table (5x5):");
for (int i = 1; i <= 5; i++) {
    for (int j = 1; j <= 5; j++) {
        System.out.printf("%4d", i * j);
    }
    System.out.println();
}

// Enhanced for loop (for-each)
int[] numbers = {10, 20, 30, 40, 50};
```

```
        System.out.println("\nFor-each loop:");
        for (int num : numbers) {
            System.out.print(num + " ");
        }

        // For-each with 2D array
        int[][] matrix = {{1, 2}, {3, 4}, {5, 6}};
        System.out.println("\n\n2D array using for-each:");
        for (int[] row : matrix) {
            for (int val : row) {
                System.out.print(val + " ");
            }
            System.out.println();
        }

        // Sum using for-each
        int sum = 0;
        for (int num : numbers) {
            sum += num;
        }
        System.out.println("\nSum: " + sum);
    }
```

```
}
```

while Loop

```
public class WhileLoopDemo {
```

```
    public static void main(String[] args) {
        // Basic while loop
        System.out.println("Numbers 1 to 10:");
        int i = 1;
        while (i <= 10) {
            System.out.print(i + " ");
            i++;
        }

        // Sum of digits
        System.out.println("\n\nSum of digits of 12345:");
        int num = 12345;
        int sum = 0;
```

```
while (num > 0) {
    sum += num % 10;
    num /= 10;
}
System.out.println("Sum = " + sum);

// Reverse a number
int number = 12345;
int reversed = 0;
int temp = number;
while (temp > 0) {
    reversed = reversed * 10 + (temp % 10);
    temp /= 10;
}
System.out.println("Original: " + number + ", Reversed: " + reversed);

// While loop with boolean flag
boolean running = true;
int counter = 0;
while (running) {
    System.out.print(counter + " ");
    counter++;
    if (counter >= 5) {
        running = false;
    }
}

// Sentinel-controlled loop
System.out.println("\n\nSentinel-controlled (0 to stop):");
java.util.Scanner sc = new java.util.Scanner(System.in);
int input;
int total = 0;
int count = 0;

System.out.println("Enter numbers (0 to stop):");
while (true) {
    input = sc.nextInt();
    if (input == 0) break;
    total += input;
    count++;
}
if (count > 0) {
    System.out.println("Average: " + (double)total / count);
```

```
    }
    sc.close();
  }
```

```
}
```

do-while Loop

```
public class DoWhileDemo {
```

```
  public static void main(String[] args) {
    // Basic do-while (executes at least once)
    System.out.println("Do-while example:");
    int i = 10;
    do {
      System.out.print(i + " ");
      i++;
    } while (i <= 5);  // Condition false, but loop ran once
    System.out.println("\nValue after loop: " + i);

    // Menu-driven program
    java.util.Scanner sc = new java.util.Scanner(System.in);
    int choice;

    do {
      System.out.println("\n=== MENU ===");
      System.out.println("1. Say Hello");
      System.out.println("2. Show Date");
      System.out.println("3. Exit");
      System.out.print("Enter choice: ");

      choice = sc.nextInt();

      switch (choice) {
        case 1:
          System.out.println("Hello!");
          break;
        case 2:
          System.out.println("Today is: " + java.time.LocalDate.now());
          break;
        case 3:
          System.out.println("Goodbye!");
```

```
                break;
            default:
                System.out.println("Invalid choice");
        }
    } while (choice != 3);

    // Password validation
    String password;
    do {
        System.out.print("\nEnter password (min 6 chars): ");
        password = sc.next();
        if (password.length() < 6) {
            System.out.println("Too short! Try again.");
        }
    } while (password.length() < 6);
    System.out.println("Password accepted!");

    sc.close();
  }
}
```

---

23. Jump Statements

Jump statements (break, continue, return) alter the normal flow of control in a program.

break Statement

The break statement is used to terminate a loop or a switch statement. Control is transferred to the statement immediately following the loop or switch.

```
public class BreakDemo {

  public static void main(String[] args) {
    // Breaking out of loop
    System.out.println("Finding first number divisible by 7:");
    for (int i = 1; i <= 100; i++) {
      if (i % 7 == 0) {
```

```
        System.out.println("Found: " + i);
        break;  // Exit the loop
    }
}

// break in nested loops (breaks inner loop only)
System.out.println("\nNested loops with break:");
for (int i = 1; i <= 3; i++) {
    for (int j = 1; j <= 3; j++) {
        if (j == 2) break;  // Breaks inner loop when j=2
        System.out.println("i=" + i + ", j=" + j);
    }
}

// Labeled break (breaks outer loop)
System.out.println("\nLabeled break:");
outer:
for (int i = 1; i <= 3; i++) {
    for (int j = 1; j <= 3; j++) {
        if (i == 2 && j == 2) {
            break outer;  // Breaks both loops
        }
        System.out.println("i=" + i + ", j=" + j);
    }
}

// break in switch
int day = 3;
switch (day) {
    case 1:
        System.out.println("Monday");
        break;  // Without break, would fall through
    case 2:
        System.out.println("Tuesday");
        break;
    case 3:
        System.out.println("Wednesday");
        break;
    default:
        System.out.println("Other day");
}

// Searching in array
```

```
        int[] numbers = {10, 20, 30, 40, 50};
        int search = 30;
        int foundIndex = -1;

        for (int i = 0; i < numbers.length; i++) {
            if (numbers[i] == search) {
                foundIndex = i;
                break;  // Stop searching once found
            }
        }

        if (foundIndex != -1) {
            System.out.println("\nFound " + search + " at index " + foundIndex);
        } else {
            System.out.println("\nNot found");
        }
    }
```

```
}
```

continue Statement

The continue statement is used to skip the current iteration of a loop and proceed to the next iteration.

```
public class ContinueDemo {
```

```
    public static void main(String[] args) {
        // Skip even numbers
        System.out.println("Odd numbers from 1 to 10:");
        for (int i = 1; i <= 10; i++) {
            if (i % 2 == 0) {
                continue;  // Skip even numbers
            }
            System.out.print(i + " ");
        }

        // Skip vowels
        System.out.println("\n\nString without vowels:");
        String text = "Hello World";
        for (char c : text.toCharArray()) {
            if ("AEIOUaeiou".indexOf(c) != -1) {
```

```
            continue;
        }
        System.out.print(c);
    }

    // continue in while loop
    System.out.println("\n\nNumbers not divisible by 3:");
    int i = 1;
    while (i <= 20) {
        if (i % 3 == 0) {
            i++;
            continue;
        }
        System.out.print(i + " ");
        i++;
    }

    // Labeled continue
    System.out.println("\n\nLabeled continue (skip inner loop iterations):");
    outer:
    for (int x = 1; x <= 3; x++) {
        for (int y = 1; y <= 3; y++) {
            if (x == 2 && y == 2) {
                continue outer;  // Skip rest of inner loop and continue outer
            }
            System.out.println("x=" + x + ", y=" + y);
        }
    }

    // Filtering array elements
    int[] numbers = {1, -2, 3, -4, 5, -6, 7};
    System.out.print("\nPositive numbers: ");
    for (int num : numbers) {
        if (num < 0) continue;
        System.out.print(num + " ");
    }
  }
```

```
}
```

return Statement

The return statement is used to exit from a method and return control to the caller. It can optionally return a value.

```
public class ReturnDemo {

    // Method with return value
    static int add(int a, int b) {
        return a + b;  // Returns value to caller
    }

    // Method with void return
    static void printMessage(String msg) {
        if (msg == null || msg.isEmpty()) {
            return;  // Early exit from void method
        }
        System.out.println("Message: " + msg);
    }

    // Method returning early based on condition
    static int divide(int a, int b) {
        if (b == 0) {
            System.out.println("Cannot divide by zero");
            return -1;  // Return error code
        }
        return a / b;
    }

    // Method returning object
    static String getGreeting(String name) {
        if (name == null) {
            return "Hello Guest";
        }
        return "Hello " + name;
    }

    // Method with multiple return points
    static int max(int a, int b, int c) {
        if (a >= b && a >= c) return a;
        if (b >= a && b >= c) return b;
        return c;
    }
```

```
    public static void main(String[] args) {
        int sum = add(10, 20);
        System.out.println("Sum: " + sum);

        printMessage("Hello");
        printMessage("");  // Returns early, nothing printed

        int result = divide(10, 2);
        System.out.println("10 / 2 = " + result);

        result = divide(10, 0);
        System.out.println("Division result: " + result);

        String greeting = getGreeting("Alice");
        System.out.println(greeting);

        greeting = getGreeting(null);
        System.out.println(greeting);

        System.out.println("Max of 10, 20, 15: " + max(10, 20, 15));
    }
}
```

Quick Reference - Module 1

Data Types Summary:

Type	Size	Range
byte	1	-128 to 127
short	2	-32,768 to 32,767
int	4	-2^{31} to $2^{31}-1$
long	8	-2^{63} to $2^{63}-1$

Type	Size	Range
float	4	±3.4e-38 to ±3.4e+38
double	8	±1.7e-308 to ±1.7e+308
char	2	0 to 65,535
boolean	1 bit	true/false

Control Statements:

Type	Keywords
Selection	if, else, switch, case, default
Iteration	for, while, do-while
Jump	break, continue, return

Common Escape Sequences:

Sequence	Meaning
\n	Newline
\t	Tab
\r	Carriage return
\b	Backspace
\'	Single quote
\"	Double quote
\\	Backslash

End of Module 1: An Overview of Java