Introduction to Java and Core OOP Concepts

Dr. Ratnesh Prasad Srivastava
Department of CSIT, GGV, Bilaspur (C.G.)

Academic Year: 2026-27

Course Information

Course Code	CIUDMJT1
Course Title	Object-Oriented Programming with Java
Credit Hours	3-0-3 (3 Lecture, 0 Tutorial, 3 Practical)
Prerequisites	Programming Fundamentals
Textbook	"Java: The Complete Reference" by Herbert Schildt
Reference	"Head First Java" by Kathy Sierra and Bert Bates

Contents

1 Unit I: Introduction to Java and Core OOP Concepts

1.1 Learning Objectives

- Understand the evolution and significance of Java programming language
- Master Java development environment setup and compilation process
- Implement programs using Java syntax, variables, and control structures
- Comprehend Object-Oriented Programming paradigm and its implementation in Java
- Apply encapsulation principles with proper access control
- Design and implement classes with methods, constructors, and overloading
- Manipulate single and multi-dimensional arrays for data organization

2 Introduction to Java

2.1 Historical Development of Java

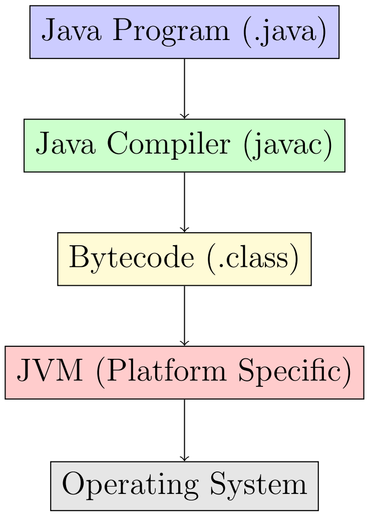
Historical Timeline

1991 - Project "Oak" started by James Gosling at Sun Microsystems
1995 - Renamed to "Java" and JDK 1.0 released
1998 - Java 2 Platform (J2SE 1.2) with Swing GUI
2004 - J2SE 5.0 introduced generics, annotations, autoboxing
2014 - Java SE 8 with Lambda expressions and Stream API
2021 - Java 17 (Long-Term Support version)
2023 - Java 21 with virtual threads and pattern matching

2.2 Key Features of Java

1. **Platform Independence:** "Write Once, Run Anywhere" (WORA) through bytecode
2. **Object-Oriented:** Everything is an object (except primitives)
3. **Simple and Familiar:** C/C++ like syntax without pointers
4. **Security:** Built-in security features for network applications
5. **Multithreading:** Built-in support for concurrent programming
6. **Distributed:** Network-centric design with RMI and CORBA

2.3 Java Platform Architecture



2.3.1 Java Development Kit (JDK) Components

- **javac:** Java compiler that converts .java to .class files
- **java:** Java interpreter that executes bytecode
- **jar:** Java archive tool for packaging classes
- **javadoc:** Documentation generator
- **jdb:** Java debugger
- **JRE:** Runtime environment for executing Java applications

2.4 Detailed Java Development Process

```java
// Line 1: Declare a public class named 'HelloWorld'
// 'public' means this class is accessible from anywhere
// 'class' keyword defines a new class
// 'HelloWorld' is the class name (must match filename)
public class HelloWorld {

    // Line 3: Main method - entry point of Java program
    // 'public' - accessible from outside
    // 'static' - can be called without creating object
    // 'void' - returns nothing
    // 'main' - mandatory method name
    // 'String[] args' - command line arguments array
    public static void main(String[] args) {

        // Line 4: System.out.println prints to console
        // 'System' is a predefined class
        // 'out' is a static member of System class (PrintStream object
           )
        // 'println' method prints string with new line
        System.out.println("Hello, Welcome to Java Programming!");

        // Line 5: Printing command line arguments count
        // 'args.length' gives number of arguments passed
        System.out.println("Number of arguments: " + args.length);

        // Line 6-9: Printing all arguments if any
        for (int i = 0; i < args.length; i++) {
            System.out.println("Argument " + (i+1) + ": " + args[i]);
        }
    }
}
```

Listing 1: Complete Development Cycle with Line-by-Line Explanation

Compilation and Execution Steps

Step 1: Save file as `HelloWorld.java`
Step 2: Compile: `javac HelloWorld.java`
Step 3: Execute: `java HelloWorld "FirstArg" "SecondArg"`
Step 4: Output:

```
Hello, Welcome to Java Programming!
Number of arguments: 2
Argument 1: FirstArg
Argument 2: SecondArg
```

3 Basic Java Syntax - In Depth

3.1 Detailed Variable Declaration and Data Types

```java
public class DataTypesDemo {
    public static void main(String[] args) {
        // ================= PRIMITIVE DATA TYPES =================
        // Line 4: byte - 8-bit signed integer (-128 to 127)
        // Used for saving memory in large arrays
        byte studentAge = 20;  // Memory: 1 byte
        System.out.println("Byte - Student Age: " + studentAge);

        // Line 7: short - 16-bit integer (-32768 to 32767)
        short totalStudents = 1500;  // Memory: 2 bytes
        System.out.println("Short - Total Students: " + totalStudents);

        // Line 10: int - 32-bit integer (default for whole numbers)
        int population = 1400000000;  // Memory: 4 bytes
        System.out.println("Int - Population: " + population);

        // Line 13: long - 64-bit integer (Use 'L' suffix for literal)
        long worldPopulation = 8000000000L;  // Memory: 8 bytes
        System.out.println("Long - World Population: " +
            worldPopulation);

        // Line 16: float - 32-bit floating point (Use 'f' suffix)
        // Single precision, 6-7 decimal digits
        float averageMarks = 85.75f;  // Memory: 4 bytes
        System.out.println("Float - Average Marks: " + averageMarks);

        // Line 19: double - 64-bit floating point (default for decimals)
        // Double precision, 15-16 decimal digits
        double preciseValue = 3.141592653589793;  // Memory: 8 bytes
        System.out.println("Double - PI Value: " + preciseValue);

        // Line 22: char - 16-bit Unicode character
        char grade = 'A';  // Memory: 2 bytes (Unicode support)
        System.out.println("Char - Grade: " + grade);

        // Line 25: boolean - true or false (not 0 or 1)
        boolean isPassed = true;  // Memory: JVM dependent
        System.out.println("Boolean - Passed: " + isPassed);

        // =============== REFERENCE DATA TYPES =================
        // Line 28: String - Sequence of characters (Immutable)
        // String is a class, not primitive
        String studentName = "John Doe";
        System.out.println("String - Student Name: " + studentName);

        // Line 31: Array - Collection of similar data types
        int[] marksArray = {85, 90, 78, 92, 88};  // Memory: variable
        System.out.println("Array - First Mark: " + marksArray[0]);

        // =============== TYPE CASTING DEMONSTRATION ===========
        // Line 34: Implicit Casting (Widening) - Automatic
        // byte    short    int    long    float    double
        byte smallNum = 100;
        int largeNum = smallNum;  // Automatic type conversion
        System.out.println("Implicit Cast - Byte to Int: " + largeNum);

        // Line 39: Explicit Casting (Narrowing) - Manual
        // double    float    long    int    short    byte
        double bigValue = 99.99;
        int intValue = (int) bigValue;  // Manual casting, loses decimal
        System.out.println("Explicit Cast - Double to Int: " + intValue);
    }
}
```

Listing 2: Complete Data Types Example with Explanations

3.2 Operators - Complete Classification with Examples

```java
public class OperatorsDemo {
    public static void main(String[] args) {
        int a = 10, b = 3;
        boolean x = true, y = false;

        // =============== ARITHMETIC OPERATORS ===============
        System.out.println("=== ARITHMETIC OPERATORS ===");
        System.out.println("a + b = " + (a + b));      // Addition: 13
        System.out.println("a - b = " + (a - b));      // Subtraction: 7
        System.out.println("a * b = " + (a * b));      // Multiplication: 30
        System.out.println("a / b = " + (a / b));      // Division: 3 (integer division)
        System.out.println("a % b = " + (a % b));      // Modulus: 1 (remainder)

        // Increment/Decrement Operators
        int c = 5;
        System.out.println("Original c: " + c);         // 5
        System.out.println("Post-increment c++: " + (c++));  // Prints 5, then becomes 6
        System.out.println("After increment c: " + c);         // 6
        System.out.println("Pre-increment ++c: " + (++c));     // Becomes 7, then prints 7

        // =============== RELATIONAL OPERATORS ===============
        System.out.println("\n=== RELATIONAL OPERATORS ===");
        System.out.println("a == b: " + (a == b));     // Equal to: false
        System.out.println("a != b: " + (a != b));     // Not equal: true
        System.out.println("a > b: " + (a > b));       // Greater than: true
        System.out.println("a < b: " + (a < b));       // Less than: false
        System.out.println("a >= b: " + (a >= b));     // Greater than or equal: true
        System.out.println("a <= b: " + (a <= b));     // Less than or equal: false

        // =============== LOGICAL OPERATORS ===============
        System.out.println("\n=== LOGICAL OPERATORS ===");
        System.out.println("x && y: " + (x && y));     // Logical AND: false
        System.out.println("x || y: " + (x || y));     // Logical OR: true
        System.out.println("!x: " + (!x));             // Logical NOT: false

        // Short-circuit evaluation demonstration
        int n = 0;
        boolean result = (n != 0) && (10/n > 1);  // Second part not evaluated
        System.out.println("Short-circuit result: " + result);  // false

        // =============== ASSIGNMENT OPERATORS ===============
        System.out.println("\n=== ASSIGNMENT OPERATORS ===");
        int d = 15;
        d += 5;  // Equivalent to d = d + 5
        System.out.println("d += 5: " + d);  // 20

        d -= 3;  // d = d - 3
        System.out.println("d -= 3: " + d);  // 17

        d *= 2;  // d = d * 2
        System.out.println("d *= 2: " + d);  // 34

        d /= 4;  // d = d / 4
        System.out.println("d /= 4: " + d);  // 8

        d %= 3;  // d = d % 3
        System.out.println("d %= 3: " + d);  // 2

        // =============== BITWISE OPERATORS ===============
        System.out.println("\n=== BITWISE OPERATORS ===");
        int num1 = 5;    // Binary: 0101
        int num2 = 3;    // Binary: 0011

        System.out.println("num1 & num2: " + (num1 & num2));  // AND: 0001 = 1
        System.out.println("num1 | num2: " + (num1 | num2));  // OR: 0111 = 7
        System.out.println("num1 ^ num2: " + (num1 ^ num2));  // XOR: 0110 = 6
        System.out.println("~num1: " + (~num1));              // NOT: ...111010 = -6
        System.out.println("num1 << 1: " + (num1 << 1));      // Left shift: 1010 = 10
        System.out.println("num1 >> 1: " + (num1 >> 1));      // Right shift: 0010 = 2

        // =============== TERNARY OPERATOR ===============
        System.out.println("\n=== TERNARY OPERATOR ===");
        int score = 75;
        String resultMsg = (score >= 40) ? "Pass" : "Fail";
        System.out.println("Score: " + score + ", Result: " + resultMsg);
    }
}
```

Listing 3: Operators in Java with Detailed Examples

Output of OperatorsDemo Program

```
=== ARITHMETIC OPERATORS ===
a + b = 13
a - b = 7
a * b = 30
a / b = 3
a % b = 1
Original c: 5
Post-increment c++: 5
After increment c: 6
Pre-increment ++c: 7

=== RELATIONAL OPERATORS ===
a == b: false
a != b: true
a > b: true
a < b: false
a >= b: true
a <= b: false

=== LOGICAL OPERATORS ===
x && y: false
x || y: true
!x: false
Short-circuit result: false

=== ASSIGNMENT OPERATORS ===
d += 5: 20
d -= 3: 17
d *= 2: 34
d /= 4: 8
d %= 3: 2

=== BITWISE OPERATORS ===
num1 & num2: 1
num1 | num2: 7
num1 ^ num2: 6
~num1: -6
num1 << 1: 10
num1 >> 1: 2

=== TERNARY OPERATOR ===
Score: 75, Result: Pass
```

4 Control Flow Statements - Detailed Implementation

4.1 If-Else Statements with Complete Example

```java
import java.util.Scanner;  // Import Scanner class for user input

public class GradeCalculator {
    public static void main(String[] args) {
        // Line 3: Create Scanner object to read input from keyboard
        Scanner scanner = new Scanner(System.in);

        // Line 5: Prompt user to enter marks
        System.out.print("Enter your marks (0-100): ");

        // Line 7: Read integer input from user
        int marks = scanner.nextInt();

        // Line 9: Validate input range
        if (marks < 0 || marks > 100) {
            System.out.println("Invalid marks! Please enter between 0
                and 100.");
            return;  // Exit program if invalid input
        }

        char grade;
        String message;

        // ============= IF-ELSE LADDER =============
        // Line 18: Check for grade A (90-100)
        if (marks >= 90) {
            grade = 'A';
            message = "Excellent! First Class with Distinction";
        }
        // Line 22: Check for grade B (80-89)
        else if (marks >= 80) {
            grade = 'B';
            message = "Very Good! First Class";
        }
        // Line 26: Check for grade C (70-79)
        else if (marks >= 70) {
            grade = 'C';
            message = "Good! Second Class";
        }
        // Line 30: Check for grade D (60-69)
        else if (marks >= 60) {
            grade = 'D';
            message = "Satisfactory";
        }
        // Line 34: Check for grade E (40-59)
        else if (marks >= 40) {
            grade = 'E';
            message = "Pass";
        }
        // Line 38: Otherwise grade F (0-39)
        else {
            grade = 'F';
            message = "Fail - Needs Improvement";
        }

        // Line 42: Display result
        System.out.println("\n========== RESULT ==========");
        System.out.println("Marks Obtained: " + marks);
        System.out.println("Grade Awarded: " + grade);
        System.out.println("Remarks: " + message);

        // Line 47: Additional performance category
        if (marks >= 75) {
            System.out.println("Performance: Outstanding!");
        } else if (marks >= 50) {
            System.out.println("Performance: Average");
        } else {
            System.out.println("Performance: Below Average");
        }

        // Line 54: Close scanner to prevent resource leak
        scanner.close();
    }
}
```

Listing 4: Grading System with If-Else Ladder

Sample Outputs for GradeCalculator

Test Case 1: Input 95

```
Enter your marks (0-100): 95

========== RESULT ==========
Marks Obtained: 95
Grade Awarded: A
Remarks: Excellent! First Class with Distinction
Performance: Outstanding!
```

Test Case 2: Input 65

```
Enter your marks (0-100): 65

========== RESULT ==========
Marks Obtained: 65
Grade Awarded: D
Remarks: Satisfactory
Performance: Average
```

Test Case 3: Input 35

```
Enter your marks (0-100): 35

========== RESULT ==========
Marks Obtained: 35
Grade Awarded: F
Remarks: Fail - Needs Improvement
Performance: Below Average
```

4.2 Switch Statement with Modern Features

```java
import java.time.DayOfWeek;
import java.time.LocalDate;

public class EnhancedSwitchDemo {
    public static void main(String[] args) {
        // Line 4: Get current day of week
        DayOfWeek today = LocalDate.now().getDayOfWeek();
        System.out.println("Today is: " + today);

        // ========== TRADITIONAL SWITCH (Pre-Java 14) ==========
        System.out.println("\n=== Traditional Switch ===");
        int dayNumber = today.getValue();  // 1=Monday, 7=Sunday

        String dayType;
        switch (dayNumber) {
            case 1:
            case 2:
            case 3:
```

```java
            case 4:
            case 5:
                dayType = "Weekday";
                break;
            case 6:
            case 7:
                dayType = "Weekend";
                break;
            default:
                dayType = "Invalid Day";
        }
        System.out.println("Day Type: " + dayType);

        // ========== ENHANCED SWITCH (Java 14+) ==========
        System.out.println("\n=== Enhanced Switch ===");

        // Switch expression with arrow syntax
        String schedule = switch (today) {
            case MONDAY -> {
                // Yield returns value from block
                yield "Team Meeting at 10 AM";
            }
            case TUESDAY, WEDNESDAY, THURSDAY -> "Regular Work Day";
            case FRIDAY -> "Casual Friday - Weekly Review";
            case SATURDAY -> "Weekend - Relax!";
            case SUNDAY -> "Weekend - Family Time";
            // No default needed for exhaustive enum
        };

        System.out.println("Today's Schedule: " + schedule);

        // ========== SWITCH WITH MULTIPLE VALUES ==========
        System.out.println("\n=== Switch with Multiple Cases ===");
        int month = LocalDate.now().getMonthValue();

        String season = switch (month) {
            case 12, 1, 2 -> "Winter";
            case 3, 4, 5 -> "Spring";
            case 6, 7, 8 -> "Summer";
            case 9, 10, 11 -> "Autumn";
            default -> "Invalid Month";
        };

        System.out.println("Current Season: " + season);

        // ========== SWITCH WITH RETURN VALUES ==========
        System.out.println("\n=== Switch Returning Values ===");

        int dayNum = 3;  // Wednesday
        String dayName = getDayName(dayNum);
        System.out.println("Day " + dayNum + " is: " + dayName);
    }

    // Method demonstrating switch with return
    public static String getDayName(int day) {
        return switch (day) {
            case 1 -> "Monday";
            case 2 -> "Tuesday";
```

```
                case 3 -> "Wednesday";
                case 4 -> "Thursday";
                case 5 -> "Friday";
                case 6 -> "Saturday";
                case 7 -> "Sunday";
                default -> "Invalid Day Number";
            };
        }
    }
```

Listing 5: Enhanced Switch Statement with Multiple Cases

4.3 Loops - Complete Implementation with Patterns

```
public class LoopPatterns {
    public static void main(String[] args) {
        System.out.println("=== FOR LOOP PATTERNS ===\n");

        // ========== PATTERN 1: Right Triangle ==========
        System.out.println("Pattern 1: Right Triangle");
        for (int i = 1; i <= 5; i++) {
            for (int j = 1; j <= i; j++) {
                System.out.print("* ");
            }
            System.out.println();
        }

        // ========== PATTERN 2: Number Pyramid ==========
        System.out.println("\nPattern 2: Number Pyramid");
        for (int i = 1; i <= 4; i++) {
            // Print spaces
            for (int j = i; j < 4; j++) {
                System.out.print("  ");
            }
            // Print numbers
            for (int j = 1; j <= (2*i - 1); j++) {
                System.out.print(j + " ");
            }
            System.out.println();
        }

        System.out.println("\n=== WHILE LOOP EXAMPLE ===\n");

        // ========== FACTORIAL USING WHILE ==========
        int number = 5;
        int factorial = 1;
        int temp = number;

        System.out.println("Calculating factorial of " + number);
        while (temp > 0) {
            factorial *= temp;
            System.out.println("Current value: " + temp +
                             ", Factorial so far: " + factorial);
            temp--;
        }
        System.out.println("Final Result: " + number +
                         "! = " + factorial);
```

```java

        System.out.println("\n=== DO-WHILE EXAMPLE ===\n");

        // ========== MENU DRIVEN PROGRAM ==========
        java.util.Scanner scanner = new java.util.Scanner(System.in);
        int choice;

        do {
            System.out.println("\n===== MENU =====");
            System.out.println("1. Print Hello");
            System.out.println("2. Print Numbers 1-5");
            System.out.println("3. Exit");
            System.out.print("Enter your choice: ");

            choice = scanner.nextInt();

            switch (choice) {
                case 1:
                    System.out.println("Hello World!");
                    break;
                case 2:
                    for (int i = 1; i <= 5; i++) {
                        System.out.print(i + " ");
                    }
                    System.out.println();
                    break;
                case 3:
                    System.out.println("Exiting program...");
                    break;
                default:
                    System.out.println("Invalid choice! Try again.");
            }
        } while (choice != 3);  // Loop until user chooses 3

        scanner.close();

        System.out.println("\n=== LOOP CONTROL STATEMENTS ===\n");

        // ========== BREAK STATEMENT ==========
        System.out.println("Break Example: Find first multiple of 7");
        for (int i = 1; i <= 50; i++) {
            if (i % 7 == 0) {
                System.out.println("First multiple of 7: " + i);
                break;  // Exit loop immediately
            }
        }

        // ========== CONTINUE STATEMENT ==========
        System.out.println("\nContinue Example: Print odd numbers 1-10"
            );
        for (int i = 1; i <= 10; i++) {
            if (i % 2 == 0) {
                continue;  // Skip even numbers
            }
            System.out.print(i + " ");
        }
        System.out.println();

        // ========== LABELED BREAK ==========
        System.out.println("\nLabeled Break Example: Matrix Search");
        outerLoop:  // Label for outer loop
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                int value = i * 3 + j;
                if (value == 5) {
                    System.out.println("Found 5 at position: (" +
                                    i + "," + j + ")");
                    break outerLoop;  // Break out of both loops
                }
            }
        }
    }
}
```

Listing 6: Loop Patterns and Applications

LoopPatterns Program Output

```
=== FOR LOOP PATTERNS ===
Pattern 1: Right Triangle
*
* *
* * *
* * * *
* * * * *

Pattern 2: Number Pyramid
      1
    1 2 3
  1 2 3 4 5
1 2 3 4 5 6 7

=== WHILE LOOP EXAMPLE ===
Calculating factorial of 5
Current value: 5, Factorial so far: 5
Current value: 4, Factorial so far: 20
Current value: 3, Factorial so far: 60
Current value: 2, Factorial so far: 120
Current value: 1, Factorial so far: 120
Final Result: 5! = 120
=== DO-WHILE EXAMPLE ===
===== MENU =====
1. Print Hello
2. Print Numbers 1-5
3. Exit
Enter your choice: 1
Hello World!
===== MENU =====
1. Print Hello
2. Print Numbers 1-5
3. Exit
Enter your choice: 2
1 2 3 4 5
===== MENU =====
1. Print Hello
2. Print Numbers 1-5
3. Exit
Enter your choice: 3
Exiting program...
=== LOOP CONTROL STATEMENTS ===
Break Example: Find first multiple of 7
First multiple of 7: 7
Continue Example: Print odd numbers 1-10
1 3 5 7 9
Labeled Break Example: Matrix Search
Found 5 at position: (1,2)
```

5 Object-Oriented Programming Fundamentals

5.1 Classes and Objects - Real World Example

```java
// ==================== BANK ACCOUNT CLASS ====================
class BankAccount {
    // ============= ATTRIBUTES (INSTANCE VARIABLES) =============
    // These define the STATE of the object
    private String accountNumber;    // Unique identifier
    private String accountHolder;    // Name of account owner
    private double balance;          // Current balance
    private String accountType;      // Savings/Current/Fixed Deposit
    private static int totalAccounts = 0;  // Class variable - shared
        by all

    // ============= CONSTRUCTOR =============
    // Special method called when object is created
    public BankAccount(String holder, String type, double
        initialDeposit) {
        // 'this' refers to current object instance
        this.accountHolder = holder;
        this.accountType = type;
        this.balance = initialDeposit;

        // Generate unique account number
        this.accountNumber = "ACC" + (1000 + ++totalAccounts);

        System.out.println("New account created: " + this.accountNumber
            );
        System.out.println("Total accounts in bank: " + totalAccounts);
    }

    // ============= BEHAVIORS (METHODS) =============
    // These define what the object can DO

    // Deposit money
    public void deposit(double amount) {
        if (amount > 0) {
            balance += amount;
            System.out.println("Deposited: $" + amount);
            System.out.println("New Balance: $" + balance);
        } else {
            System.out.println("Invalid deposit amount!");
        }
    }

    // Withdraw money with validation
    public boolean withdraw(double amount) {
        if (amount > 0 && amount <= balance) {
            balance -= amount;
            System.out.println("Withdrawn: $" + amount);
            System.out.println("Remaining Balance: $" + balance);
            return true;  // Successful withdrawal
        } else {
            System.out.println("Withdrawal failed! Insufficient funds
                or invalid amount");
            return false;  // Failed withdrawal
        }
    }

    // Check balance
    public void checkBalance() {
        System.out.println("Account: " + accountNumber);
        System.out.println("Holder: " + accountHolder);
        System.out.println("Balance: $" + balance);
        System.out.println("Account Type: " + accountType);
    }

    // Transfer money to another account
    public void transfer(BankAccount recipient, double amount) {
        System.out.println("\n=== Initiating Transfer ===");
        System.out.println("From: " + this.accountHolder);
        System.out.println("To: " + recipient.accountHolder);
        System.out.println("Amount: $" + amount);

        if (this.withdraw(amount)) {
            recipient.deposit(amount);
            System.out.println("Transfer completed successfully!");
        } else {
            System.out.println("Transfer failed!");
        }
    }

    // ============= GETTER METHODS =============
    // Provide controlled access to private data
    public double getBalance() {
        return balance;
    }

    public String getAccountNumber() {
        return accountNumber;
    }

    public String getAccountHolder() {
        return accountHolder;
    }
}

// ==================== MAIN CLASS ====================
public class BankingSystem {
    public static void main(String[] args) {
        System.out.println("=== WELCOME TO JAVA BANK ===\n");

        // ============= CREATING OBJECTS =============
        // Line 79-81: Creating account1 object
        // 'new' keyword allocates memory and calls constructor
        BankAccount account1 = new BankAccount("John Smith",
                                               "Savings", 1000.0);

        // Line 83-84: Creating account2 object
        BankAccount account2 = new BankAccount("Alice Johnson",
                                               "Current", 5000.0);

        System.out.println("\n=== ACCOUNT OPERATIONS ===\n");

        // ============= OPERATIONS ON ACCOUNT1 =============
        System.out.println("--- Account 1 Operations ---");
        account1.checkBalance();  // Call method on account1 object

        account1.deposit(500.0);  // Deposit money
        account1.withdraw(200.0); // Withdraw money

        // ============= OPERATIONS ON ACCOUNT2 =============
        System.out.println("\n--- Account 2 Operations ---");
        account2.checkBalance();

        account2.deposit(1000.0);
        account2.withdraw(7000.0);  // This should fail

        // ============= TRANSFER BETWEEN ACCOUNTS =============
        System.out.println("\n=== INTER-ACCOUNT TRANSFER ===");
        account1.transfer(account2, 300.0);

        // ============= FINAL BALANCES =============
        System.out.println("\n=== FINAL ACCOUNT STATUS ===");
        System.out.println("Account 1 Balance: $" + account1.getBalance
            ());
        System.out.println("Account 2 Balance: $" + account2.getBalance
            ());

        // ============= DEMONSTRATING OBJECT IDENTITY =============
        System.out.println("\n=== OBJECT COMPARISON ===");
        BankAccount account3 = account1;  // Reference copy, not object
             copy

        System.out.println("account1 == account3: " + (account1 ==
            account3));
        System.out.println("account1 == account2: " + (account1 ==
            account2));

        System.out.println("\n=== BANK STATISTICS ===");
        System.out.println("Total accounts created: " +
                                    BankAccount.totalAccounts);
    }
}
```

Listing 7: Banking System with Classes and Objects

BankingSystem Program Output

```
=== WELCOME TO JAVA BANK ===

New account created: ACC1001
Total accounts in bank: 1
New account created: ACC1002
Total accounts in bank: 2

=== ACCOUNT OPERATIONS ===

--- Account 1 Operations ---
Account: ACC1001
Holder: John Smith
Balance: $1000.0
Account Type: Savings
Deposited: $500.0
New Balance: $1500.0
Withdrawn: $200.0
Remaining Balance: $1300.0

--- Account 2 Operations ---
Account: ACC1002
Holder: Alice Johnson
Balance: $5000.0
Account Type: Current
Deposited: $1000.0
New Balance: $6000.0
Withdrawal failed! Insufficient funds or invalid amount

=== INTER-ACCOUNT TRANSFER ===

=== Initiating Transfer ===
From: John Smith
To: Alice Johnson
Amount: $300.0
Withdrawn: $300.0
Remaining Balance: $1000.0
Deposited: $300.0
New Balance: $6300.0
Transfer completed successfully!

=== FINAL ACCOUNT STATUS ===
Account 1 Balance: $1000.0
Account 2 Balance: $6300.0

=== OBJECT COMPARISON ===
account1 == account3: true
account1 == account2: false

=== BANK STATISTICS ===
Total accounts created: 2
```

6 Encapsulation and Access Control

6.1 Complete Encapsulation Implementation

```java
// ==================== EMPLOYEE CLASS ====================
class Employee {
    // ============= PRIVATE DATA MEMBERS =============
    // Complete data hiding - direct access not allowed
    private String employeeId;
    private String name;
    private double salary;
    private String department;
    private String email;
    private String phone;
    private static final String COMPANY = "Tech Solutions Inc.";
    private static int employeeCount = 0;

    // ============= CONSTRUCTORS =============
    // Parameterized constructor
    public Employee(String name, double salary, String department) {
        this.employeeId = "EMP" + String.format("%04d", ++employeeCount
            );
        this.name = name;
        this.salary = salary;
        this.department = department;
        this.email = name.toLowerCase().replace(" ", ".") + "@company.
            com";
        System.out.println("Employee " + this.employeeId + " created.")
            ;
    }

    // ============= GETTER METHODS =============
    // Controlled access to read private data
    public String getEmployeeId() {
        return employeeId;
    }

    public String getName() {
        return name;
    }

    public double getSalary() {
        return salary;
    }

    public String getDepartment() {
        return department;
    }

    public String getEmail() {
        return email;
    }

    public static String getCompany() {
        return COMPANY;
    }

    public static int getEmployeeCount() {
        return employeeCount;
    }

    // ============= SETTER METHODS =============
    // Controlled access to modify private data with validation

    public void setName(String name) {
        if (name != null && !name.trim().isEmpty()) {
            this.name = name;
            // Update email when name changes
            this.email = name.toLowerCase().replace(" ", ".") + "@company.com";
            System.out.println("Name updated. New email: " + this.email);
        } else {
            System.out.println("Invalid name provided!");
        }
    }

    public void setSalary(double salary) {
        if (salary >= 0) {
            double oldSalary = this.salary;
            this.salary = salary;
            System.out.println("Salary updated from $" + oldSalary +
                             " to $" + salary);
        } else {
            System.out.println("Invalid salary amount!");
        }
    }

    public void setDepartment(String department) {
        if (department != null && !department.trim().isEmpty()) {
            this.department = department;
            System.out.println("Department updated to: " + department);
        }
    }

    public void setPhone(String phone) {
        // Basic phone number validation
        if (phone != null && phone.matches("\\d{10}")) {
            this.phone = phone;
            System.out.println("Phone number updated.");
        } else {
            System.out.println("Invalid phone number! Must be 10 digits.");
        }
    }

    // ============= BUSINESS METHODS =============
    public void applyRaise(double percentage) {
        if (percentage > 0 && percentage <= 50) {  // Max 50% raise
            double raiseAmount = salary * (percentage / 100);
            salary += raiseAmount;
            System.out.println("Applied " + percentage + "% raise.");
            System.out.println("Raise amount: $" + raiseAmount);
            System.out.println("New salary: $" + salary);
        } else {
            System.out.println("Invalid raise percentage!");
        }
    }

    public void displayEmployeeInfo() {
        System.out.println("\n=== EMPLOYEE INFORMATION ===");
        System.out.println("Employee ID: " + employeeId);
        System.out.println("Name: " + name);
        System.out.println("Department: " + department);
        System.out.println("Email: " + email);
        System.out.println("Phone: " + (phone != null ? phone : "Not provided"));
        System.out.println("Salary: $" + salary);
        System.out.println("Company: " + COMPANY);
    }

    // ============= STATIC UTILITY METHOD =============
    public static double calculateAnnualSalary(double monthlySalary) {
        return monthlySalary * 12;
    }
}

// ==================== ACCESS MODIFIER DEMO ====================
class Department {
    // Demonstrating all access modifiers
    public String deptName;          // Accessible from anywhere
    String location;                 // Default - accessible within package
    protected int employeeCount;     // Accessible within package + subclasses
    private double budget;          // Accessible only within this class

    public Department(String name, String loc) {
        this.deptName = name;
        this.location = loc;
        this.budget = 100000.0;
    }

    // Public method to access private budget
    public double getBudget() {
        return budget;
    }

    // Protected method
    protected void setEmployeeCount(int count) {
        this.employeeCount = count;
    }
}

// Subclass demonstrating protected access
class ITDepartment extends Department {
    public ITDepartment() {
        super("IT Department", "Floor 3");
        // Can access protected member from parent
        this.employeeCount = 50;

        // Can access public members
        System.out.println("Dept: " + this.deptName);

        // Can access default within same package
        System.out.println("Location: " + this.location);

        // CANNOT access private budget directly
        // System.out.println(this.budget);  // Compilation error

        // Must use public getter
        System.out.println("Budget: $" + getBudget());
    }
}

// ==================== MAIN CLASS ====================
public class EmployeeManagementSystem {
    public static void main(String[] args) {
        System.out.println("=== EMPLOYEE MANAGEMENT SYSTEM ===\n");
        System.out.println("Company: " + Employee.getCompany());

        // ============= CREATING ENCAPSULATED EMPLOYEES =============
        Employee emp1 = new Employee("Rajesh Kumar", 50000, "
           Development");
        Employee emp2 = new Employee("Priya Sharma", 45000, "Testing");

        System.out.println("\nTotal Employees: " + Employee.
           getEmployeeCount());

        // ============= ACCESSING DATA THROUGH GETTERS =============
        System.out.println("\n=== INITIAL EMPLOYEE DATA ===");
        System.out.println("Emp1 ID: " + emp1.getEmployeeId());
        System.out.println("Emp1 Name: " + emp1.getName());
        System.out.println("Emp1 Salary: $" + emp1.getSalary());

        // ============= MODIFYING DATA THROUGH SETTERS =============
        System.out.println("\n=== UPDATING EMPLOYEE DATA ===");

        // Valid updates
        emp1.setSalary(55000);
        emp1.setName("Rajesh Kumar Sharma");
        emp1.setDepartment("Senior Development");
        emp1.setPhone("9876543210");

        // Invalid updates (will show error messages)
        emp2.setSalary(-5000);  // Invalid salary
        emp2.setPhone("123");   // Invalid phone
        emp2.setName("");       // Invalid name

        // ============= BUSINESS OPERATIONS =============
        System.out.println("\n=== PERFORMANCE REVIEW ===");
        emp1.applyRaise(15);    // Valid raise
        emp2.applyRaise(60);    // Invalid raise (exceeds 50%)

        // ============= DISPLAY INFORMATION =============
        emp1.displayEmployeeInfo();
        emp2.displayEmployeeInfo();

        // ============= STATIC METHOD DEMONSTRATION =============
        System.out.println("\n=== FINANCIAL CALCULATIONS ===");
        double annualSalary = Employee.calculateAnnualSalary(emp1.
           getSalary());
        System.out.println("Annual Salary of " + emp1.getName() +
                          ": $" + annualSalary);

        // ============= ACCESS MODIFIER DEMONSTRATION =============
        System.out.println("\n=== DEPARTMENT ACCESS DEMO ===");
        ITDepartment itDept = new ITDepartment();

        // Can access public members
        System.out.println("Dept Name (public): " + itDept.deptName);

        // Cannot access default from different context (would error if
            in different package)
        // System.out.println(itDept.location);  // Only works in same
           package

        // Cannot access private members
        // System.out.println(itDept.budget);  // Compilation error

        // Can access protected through inheritance
        System.out.println("Employee Count (protected): " + itDept.
           employeeCount);
    }
}
```

Listing 8: Employee Management System with Full Encapsulation

EmployeeManagementSystem Program Output

```
=== EMPLOYEE MANAGEMENT SYSTEM ===
Company: Tech Solutions Inc.
Employee EMP0001 created.
Employee EMP0002 created.
Total Employees: 2
=== INITIAL EMPLOYEE DATA ===
Emp1 ID: EMP0001
Emp1 Name: Rajesh Kumar
Emp1 Salary: $50000.0
=== UPDATING EMPLOYEE DATA ===
Salary updated from $50000.0 to $55000.0
Name updated. New email: rajesh.kumar.sharma@company.com
Department updated to: Senior Development
Phone number updated.
Invalid salary amount!
Invalid phone number! Must be 10 digits.
Invalid name provided!
=== PERFORMANCE REVIEW ===
Applied 15.0% raise.
Raise amount: $8250.0
New salary: $63250.0
Invalid raise percentage!
=== EMPLOYEE INFORMATION ===
Employee ID: EMP0001
Name: Rajesh Kumar Sharma
Department: Senior Development
Email: rajesh.kumar.sharma@company.com
Phone: 9876543210
Salary: $63250.0
Company: Tech Solutions Inc.
=== EMPLOYEE INFORMATION ===
Employee ID: EMP0002
Name: Priya Sharma
Department: Testing
Email: priya.sharma@company.com
Phone: null
Salary: $45000.0
Company: Tech Solutions Inc.
=== FINANCIAL CALCULATIONS ===
Annual Salary of Rajesh Kumar Sharma: $759000.0
=== DEPARTMENT ACCESS DEMO ===
Dept: IT Department
Location: Floor 3
Budget: $100000.0
Dept Name (public): IT Department
Employee Count (protected): 50
```

7 Methods and Constructors

7.1 Method Overloading and Constructor Chaining

```java
// ==================== CALCULATOR CLASS ====================
class Calculator {
    private String model;
    private String color;
    private double memory;

    // ============= CONSTRUCTOR OVERLOADING =============
    // Default constructor
    public Calculator() {
        this("Basic", "Black", 0.0);  // Constructor chaining
        System.out.println("Default constructor called");
    }

    // Parameterized constructor with model only
    public Calculator(String model) {
        this(model, "Silver", 0.0);
        System.out.println("Single parameter constructor called");
    }

    // Parameterized constructor with all parameters
    public Calculator(String model, String color, double memory) {
        this.model = model;
        this.color = color;
        this.memory = memory;
        System.out.println("Full parameter constructor called");
        System.out.println("Created: " + color + " " + model +
                         " Calculator with " + memory + "MB memory");
    }

    // ============= METHOD OVERLOADING =============
    // All methods named 'add' but with different parameters

    // Add two integers
    public int add(int a, int b) {
        System.out.println("Adding two integers: " + a + " + " + b);
        return a + b;
    }

    // Add three integers
    public int add(int a, int b, int c) {
        System.out.println("Adding three integers: " + a + " + " +
                         b + " + " + c);
        return a + b + c;
    }

    // Add two double values
    public double add(double a, double b) {
        System.out.println("Adding two doubles: " + a + " + " + b);
        return a + b;
    }

    // Add an array of integers
    public int add(int[] numbers) {
        System.out.print("Adding array of integers: ");
        int sum = 0;
        for (int num : numbers) {
            System.out.print(num + " ");
            sum += num;
        }
        System.out.println();
        return sum;
    }

    // Add variable number of integers (varargs)
    public int add(int... numbers) {
        System.out.print("Adding variable arguments: ");
        int sum = 0;
        for (int num : numbers) {
            System.out.print(num + " ");
            sum += num;
        }
        System.out.println();
        return sum;
    }

    // Add two strings (concatenation)
    public String add(String a, String b) {
        System.out.println("Concatenating strings: \"" + a + "\" + \""
            + b + "\"");
        return a + b;
    }

    // ============= 'THIS' KEYWORD DEMONSTRATION =============
    public void setModel(String model) {
        // 'this' distinguishes instance variable from parameter
        this.model = model;
    }

    public Calculator getCurrentObject() {
        // 'this' returns current object reference
        return this;
    }

    // Method demonstrating method chaining
    public Calculator setColor(String color) {
        this.color = color;
        return this;  // Enables method chaining
    }

    public Calculator setMemory(double memory) {
        this.memory = memory;
        return this;
    }

    public void displayInfo() {
        System.out.println("Calculator [Model: " + model +
                           ", Color: " + color +
                           ", Memory: " + memory + "MB]");
    }

    // ============= STATIC METHODS =============
    // Static method - can be called without creating object
    public static double power(double base, double exponent) {
        return Math.pow(base, exponent);
    }

    public static int factorial(int n) {
        if (n <= 1) return 1;
        return n * factorial(n - 1);
    }
}

// ==================== GEOMETRY CLASS ====================
class Geometry {
    // Method overloading for area calculation

    // Area of circle
    public double calculateArea(double radius) {
        System.out.println("Calculating area of circle with radius: " +
            radius);
        return Math.PI * radius * radius;
    }

    // Area of rectangle
    public double calculateArea(double length, double width) {
        System.out.println("Calculating area of rectangle: " +
                         length + " x " + width);
        return length * width;
    }

    // Area of triangle
    public double calculateArea(double base, double height, String
       shape) {
        if (shape.equalsIgnoreCase("triangle")) {
            System.out.println("Calculating area of triangle: base=" +
                             base + ", height=" + height);
            return 0.5 * base * height;
        }
        return 0;
    }

    // Area of square (using method overloading with different
       parameter types)
    public double calculateArea(int side) {
        System.out.println("Calculating area of square with side: " +
           side);
        return side * side;
    }
}

// ==================== MAIN CLASS ====================
public class MethodOverloadingDemo {
    public static void main(String[] args) {
        System.out.println("=== METHOD OVERLOADING AND CONSTRUCTORS
           ===\n");

        // ============ CONSTRUCTOR DEMONSTRATION ============
        System.out.println("--- Constructor Overloading ---");
        Calculator calc1 = new Calculator();  // Default constructor
        System.out.println();

        Calculator calc2 = new Calculator("Scientific");  // Single
            parameter
        System.out.println();

        Calculator calc3 = new Calculator("Graphing", "Blue", 256);  //
             Full
        System.out.println();

        // ============= METHOD OVERLOADING DEMONSTRATION =============
        System.out.println("--- Method Overloading Examples ---\n");

        // Different number of parameters
        System.out.println("Result 1: " + calc3.add(10, 20));
        System.out.println("Result 2: " + calc3.add(10, 20, 30));
        System.out.println();

        // Different types of parameters
        System.out.println("Result 3: " + calc3.add(5.5, 3.2));
        System.out.println("Result 4: " + calc3.add("Hello", " World"))
            ;
        System.out.println();

        // Array parameter
        int[] numbers = {1, 2, 3, 4, 5};
        System.out.println("Result 5: " + calc3.add(numbers));
        System.out.println();

        // Varargs parameter
        System.out.println("Result 6: " + calc3.add(1, 2, 3, 4, 5, 6));
        System.out.println();

        // ============= 'THIS' KEYWORD DEMONSTRATION =============
        System.out.println("--- 'this' Keyword Usage ---");

        // Method chaining using 'this'
        calc1.setColor("Red").setMemory(128).displayInfo();

        // Getting current object reference
        Calculator currentCalc = calc1.getCurrentObject();
        System.out.println("Same object? " + (calc1 == currentCalc));
        System.out.println();

        // ============= STATIC METHOD CALLS =============
        System.out.println("--- Static Methods ---");
        System.out.println("2^8 = " + Calculator.power(2, 8));
        System.out.println("5! = " + Calculator.factorial(5));
        System.out.println();

        // ============= GEOMETRY EXAMPLE =============
        System.out.println("--- Geometry Calculations ---");
        Geometry geometry = new Geometry();

        System.out.println("Area of circle: " +
                          geometry.calculateArea(7.0));
        System.out.println("Area of rectangle: " +
                          geometry.calculateArea(5.0, 3.0));
        System.out.println("Area of triangle: " +
                           geometry.calculateArea(6.0, 4.0, "triangle"));
        System.out.println("Area of square: " +
                           geometry.calculateArea(4));

        // ============= COMPLEX METHOD CALLS =============
        System.out.println("\n--- Complex Calculations ---");

        // Nested method calls
        double result = Calculator.power(
            calc3.add(2.5, 3.5),  // 6.0
            2                     // squared
        );
        System.out.println("(2.5 + 3.5)^2 = " + result);

        // Using return value in another method call
        int sum1 = calc3.add(1, 2, 3);
        int sum2 = calc3.add(4, 5, 6);
        System.out.println("Total sum: " + calc3.add(sum1, sum2));
    }
}
```

Listing 9: Calculator Class with Method Overloading and Constructors

MethodOverloadingDemo Program Output

```
=== METHOD OVERLOADING AND CONSTRUCTORS ===
--- Constructor Overloading ---
Full parameter constructor called
Created: Black Basic Calculator with 0.0MB memory
Default constructor called

Full parameter constructor called
Created: Silver Scientific Calculator with 0.0MB memory
Single parameter constructor called
Full parameter constructor called
Created: Blue Graphing Calculator with 256.0MB memory

--- Method Overloading Examples ---
Adding two integers: 10 + 20
Result 1: 30
Adding three integers: 10 + 20 + 30
Result 2: 60
Adding two doubles: 5.5 + 3.2
Result 3: 8.7
Concatenating strings: "Hello" + " World"
Result 4: Hello World
Adding array of integers: 1 2 3 4 5
Result 5: 15
Adding variable arguments: 1 2 3 4 5 6
Result 6: 21
--- 'this' Keyword Usage ---
Calculator [Model: Basic, Color: Red, Memory: 128.0MB]
Same object? true
--- Static Methods ---
2^8 = 256.0
5! = 120
--- Geometry Calculations ---
Calculating area of circle with radius: 7.0
Area of circle: 153.93804002589985
Calculating area of rectangle: 5.0 x 3.0
Area of rectangle: 15.0
Calculating area of triangle: base=6.0, height=4.0
Area of triangle: 12.0
Calculating area of square with side: 4
Area of square: 16.0
--- Complex Calculations ---
Adding two doubles: 2.5 + 3.5
(2.5 + 3.5)^2 = 36.0
Adding three integers: 1 + 2 + 3
Adding three integers: 4 + 5 + 6
Adding two integers: 6 + 15
Total sum: 21
```

8 Arrays in Java - Complete Guide

8.1 Single-Dimensional Arrays

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

public class ArrayOperations {
    public static void main(String[] args) {
        System.out.println("=== SINGLE-DIMENSIONAL ARRAYS ===\n");

        // ============= ARRAY DECLARATION METHODS =============

        // Method 1: Declaration and separate initialization
        int[] arr1;  // Declaration only
        arr1 = new int[5];  // Memory allocation for 5 integers

        // Method 2: Declaration with initialization
        int[] arr2 = new int[5];  // All elements initialized to 0

        // Method 3: Declaration with explicit values
        int[] arr3 = {10, 20, 30, 40, 50};  // Size automatically
            determined

        // Method 4: Using array literal
        int[] arr4 = new int[]{5, 4, 3, 2, 1};

        System.out.println("Array declaration methods demonstrated.");

        // ============= BASIC ARRAY OPERATIONS =============
        System.out.println("\n--- Basic Array Operations ---");

        // Initializing array with values
        for (int i = 0; i < arr1.length; i++) {
            arr1[i] = (i + 1) * 10;  // 10, 20, 30, 40, 50
        }

        // Accessing and displaying elements
        System.out.print("Array elements: ");
        for (int i = 0; i < arr1.length; i++) {
            System.out.print(arr1[i] + " ");
        }
        System.out.println();

        // Array length property
        System.out.println("Array length: " + arr1.length);

        // ============= ARRAY TRAVERSAL METHODS =============
        System.out.println("\n--- Array Traversal Methods ---");

        // Method 1: Standard for loop
        System.out.print("Standard for loop: ");
        for (int i = 0; i < arr3.length; i++) {
            System.out.print(arr3[i] + " ");
        }
        System.out.println();

        // Method 2: Enhanced for loop (for-each)
        System.out.print("Enhanced for loop: ");
        for (int num : arr3) {
            System.out.print(num + " ");
        }
        System.out.println();

        // Method 3: Using Arrays.toString()
        System.out.println("Arrays.toString(): " + Arrays.toString(arr3
            ));

        // ============= ARRAY UTILITY METHODS =============
        System.out.println("\n--- Array Utility Methods ---");

        // Copying arrays
        int[] copyArr = Arrays.copyOf(arr3, arr3.length);
        System.out.println("Copied array: " + Arrays.toString(copyArr))
            ;

        // Sorting array
        int[] unsorted = {45, 12, 89, 34, 67};
        Arrays.sort(unsorted);
        System.out.println("Sorted array: " + Arrays.toString(unsorted)
            );

        // Searching in sorted array
        int key = 34;
        int index = Arrays.binarySearch(unsorted, key);
        System.out.println(key + " found at index: " + index);

        // Filling array with specific value
        int[] filledArr = new int[5];
        Arrays.fill(filledArr, 7);
        System.out.println("Filled array: " + Arrays.toString(filledArr
            ));

        // Comparing arrays
        System.out.println("Arrays equal? " + Arrays.equals(arr3,
            copyArr));

        // ============= PRACTICAL ARRAY EXAMPLES =============
        System.out.println("\n--- Practical Array Examples ---");

        // Example 1: Find maximum and minimum
        int[] numbers = {23, 45, 12, 67, 89, 34, 56};
        int max = numbers[0];
        int min = numbers[0];

        for (int num : numbers) {
            if (num > max) max = num;
            if (num < min) min = num;
        }
        System.out.println("Maximum: " + max + ", Minimum: " + min);

        // Example 2: Calculate average
        double sum = 0;
        for (int num : numbers) {
            sum += num;
        }
        double average = sum / numbers.length;
        System.out.printf("Average: %.2f\n", average);

        // Example 3: Reverse array
        System.out.print("Original: ");
        for (int num : numbers) System.out.print(num + " ");

        for (int i = 0; i < numbers.length / 2; i++) {
            int temp = numbers[i];
            numbers[i] = numbers[numbers.length - 1 - i];
            numbers[numbers.length - 1 - i] = temp;
        }

        System.out.print("\nReversed: ");
        for (int num : numbers) System.out.print(num + " ");
        System.out.println();

        // Example 4: Find frequency of element
        int[] freqArray = {1, 2, 3, 2, 1, 3, 4, 2, 1, 1};
        int searchElement = 2;
        int frequency = 0;

        for (int num : freqArray) {
            if (num == searchElement) {
                frequency++;
            }
        }
        System.out.println("Frequency of " + searchElement + ": " +
            frequency);

        // ============= DYNAMIC ARRAY INPUT =============
        System.out.println("\n--- Dynamic Array Input ---");
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter number of elements: ");
        int n = scanner.nextInt();

        int[] dynamicArray = new int[n];

        System.out.println("Enter " + n + " elements:");
        for (int i = 0; i < n; i++) {
            System.out.print("Element " + (i + 1) + ": ");
            dynamicArray[i] = scanner.nextInt();
        }

        System.out.println("You entered: " + Arrays.toString(
            dynamicArray));
        scanner.close();

        // ============= ARRAY OF OBJECTS =============
        System.out.println("\n--- Array of Objects ---");

        String[] names = {"Alice", "Bob", "Charlie", "Diana"};

        System.out.println("Names array:");
        for (String name : names) {
            System.out.println(name);
        }

        // Modifying array of objects
        names[1] = "Robert";
        System.out.println("After modification: " + Arrays.toString(
            names));
    }
}
```

Listing 10: Comprehensive Array Operations

8.2 Multi-Dimensional Arrays

```java
public class MultiDimensionalArrays {
    public static void main(String[] args) {
        System.out.println("=== MULTI-DIMENSIONAL ARRAYS ===\n");

        // ============= 2D ARRAYS (MATRICES) =============
        System.out.println("--- 2D Arrays / Matrices ---");

        // Method 1: Declaration with sizes
        int[][] matrix1 = new int[3][3];  // 3x3 matrix

        // Initialize with nested loops
        int value = 1;
        for (int i = 0; i < matrix1.length; i++) {
            for (int j = 0; j < matrix1[i].length; j++) {
                matrix1[i][j] = value++;
            }
        }

        // Display matrix
        System.out.println("Matrix 1 (3x3):");
        for (int i = 0; i < matrix1.length; i++) {
            for (int j = 0; j < matrix1[i].length; j++) {
                System.out.print(matrix1[i][j] + "\t");
            }
            System.out.println();
        }

        // Method 2: Declaration with initialization
        int[][] matrix2 = {
            {1, 2, 3},
            {4, 5, 6},
            {7, 8, 9}
        };

        System.out.println("\nMatrix 2 (3x3):");
        for (int[] row : matrix2) {
            for (int element : row) {
                System.out.print(element + "\t");
            }
            System.out.println();
        }

        // ============= MATRIX OPERATIONS =============
        System.out.println("\n--- Matrix Operations ---");

        // Matrix addition
        System.out.println("Matrix Addition (A + B):");
        int[][] A = {{1, 2}, {3, 4}};
        int[][] B = {{5, 6}, {7, 8}};
        int[][] C = new int[2][2];

        for (int i = 0; i < A.length; i++) {
            for (int j = 0; j < A[i].length; j++) {
                C[i][j] = A[i][j] + B[i][j];
            }
        }

        // Display result
        for (int[] row : C) {
            for (int element : row) {
                System.out.print(element + "\t");
            }
            System.out.println();
        }

        // Matrix multiplication
        System.out.println("\nMatrix Multiplication (A   B):");
        int[][] D = {{1, 2, 3}, {4, 5, 6}};  // 2x3
        int[][] E = {{7, 8}, {9, 10}, {11, 12}};  // 3x2
        int[][] F = new int[2][2];  // Result will be 2x2

        for (int i = 0; i < 2; i++) {
            for (int j = 0; j < 2; j++) {
                for (int k = 0; k < 3; k++) {
                    F[i][j] += D[i][k] * E[k][j];
                }
            }
        }

        System.out.println("Result:");
        for (int[] row : F) {
            for (int element : row) {
                System.out.print(element + "\t");
            }
            System.out.println();
        }

        // ============= JAGGED ARRAYS =============
        System.out.println("\n--- Jagged Arrays ---");

        // Creating jagged array
        int[][] jagged = new int[3][];
        jagged[0] = new int[2];  // Row 0 has 2 columns
        jagged[1] = new int[4];  // Row 1 has 4 columns
        jagged[2] = new int[3];  // Row 2 has 3 columns

        // Initialize jagged array
        int counter = 1;
        for (int i = 0; i < jagged.length; i++) {
            for (int j = 0; j < jagged[i].length; j++) {
                jagged[i][j] = counter++;
            }
        }

        // Display jagged array
        System.out.println("Jagged Array:");
        for (int i = 0; i < jagged.length; i++) {
            System.out.print("Row " + i + " (" + jagged[i].length + "
                elements): ");
            for (int j = 0; j < jagged[i].length; j++) {
                System.out.print(jagged[i][j] + " ");
            }
            System.out.println();
        }

        // ============= 3D ARRAYS =============
        System.out.println("\n--- 3D Arrays ---");

        // Creating 3D array (2x3x4)
        int[][][] threeDArray = new int[2][3][4];

        // Initialize 3D array
        int val = 1;
        for (int i = 0; i < 2; i++) {
            for (int j = 0; j < 3; j++) {
                for (int k = 0; k < 4; k++) {
                    threeDArray[i][j][k] = val++;
                }
            }
        }

        // Display 3D array
        System.out.println("3D Array (2x3x4):");
        for (int i = 0; i < 2; i++) {
            System.out.println("Layer " + i + ":");
            for (int j = 0; j < 3; j++) {
                for (int k = 0; k < 4; k++) {
                    System.out.printf("%3d ", threeDArray[i][j][k]);
                }
                System.out.println();
            }
            System.out.println();
        }

        // ============= PRACTICAL APPLICATIONS =============
        System.out.println("--- Practical Applications ---");

        // Student Marks System
        System.out.println("\nStudent Marks System:");

        // 5 students, 3 subjects each
        int[][] marks = {
            {85, 90, 88},  // Student 1
            {78, 92, 85},  // Student 2
            {90, 85, 92},  // Student 3
            {65, 70, 75},  // Student 4
            {88, 92, 90}   // Student 5
        };

        String[] subjects = {"Math", "Science", "English"};
        String[] students = {"Alice", "Bob", "Charlie", "Diana", "Eve"
            };

        // Display marks table
        System.out.print("Student\t\t");
        for (String subject : subjects) {
            System.out.print(subject + "\t");
        }
        System.out.println("Total\tAverage");
        System.out.println("=".repeat(60));

        for (int i = 0; i < marks.length; i++) {
            int total = 0;
            System.out.print(students[i] + "\t\t");

            for (int j = 0; j < marks[i].length; j++) {
                System.out.print(marks[i][j] + "\t");
                total += marks[i][j];
            }

            double average = total / (double) marks[i].length;
            System.out.printf("%d\t%.2f\n", total, average);
        }

        // Subject-wise analysis
        System.out.println("\nSubject-wise Analysis:");
        for (int j = 0; j < subjects.length; j++) {
            int subjectTotal = 0;
            for (int i = 0; i < marks.length; i++) {
                subjectTotal += marks[i][j];
            }
            double subjectAverage = subjectTotal / (double) marks.
                length;
            System.out.printf("%s: Average = %.2f\n", subjects[j],
                subjectAverage);
        }

        // ============= ARRAY MANIPULATION METHODS =============
        System.out.println("\n--- Array Manipulation Methods ---");

        // Transpose of a matrix
        System.out.println("Matrix Transpose:");
        int[][] original = {{1, 2, 3}, {4, 5, 6}};
        int rows = original.length;
        int cols = original[0].length;

        int[][] transpose = new int[cols][rows];

        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                transpose[j][i] = original[i][j];
            }
        }

        System.out.println("Original (2x3):");
        for (int[] row : original) {
            for (int element : row) {
                System.out.print(element + " ");
            }
            System.out.println();
        }

        System.out.println("Transpose (3x2):");
        for (int[] row : transpose) {
            for (int element : row) {
                System.out.print(element + " ");
            }
            System.out.println();
        }
    }
}
```

Listing 11: Multi-dimensional Arrays and Matrices

MultiDimensionalArrays Program Output (Partial)

```
=== MULTI-DIMENSIONAL ARRAYS ===
--- 2D Arrays / Matrices ---
Matrix 1 (3x3):
1 2 3
4 5 6
7 8 9
Matrix 2 (3x3):
1 2 3
4 5 6
7 8 9
--- Matrix Operations ---
Matrix Addition (A + B):
6 8
10 12
Matrix Multiplication (A × B):
Result:
58 64
139 154
--- Jagged Arrays ---
Jagged Array:
Row 0 (2 elements): 1 2
Row 1 (4 elements): 3 4 5 6
Row 2 (3 elements): 7 8 9
--- 3D Arrays ---
3D Array (2x3x4):
Layer 0:
  1   2   3   4
  5   6   7   8
  9  10  11  12
Layer 1:
 13  14  15  16
 17  18  19  20
 21  22  23  24
--- Practical Applications ---
Student Marks System:
Student  Math Science English Total Average
===================================================
Alice  85 90 88 263 87.67
Bob  78 92 85 255 85.00
Charlie 90 85 92 267 89.00
Diana  65 70 75 210 70.00
Eve  88 92 90 270 90.00
Subject-wise Analysis:
Math: Average = 81.20
Science: Average = 85.80
English: Average = 86.00
--- Array Manipulation Methods -Matrix Transpose:
Original (2x3):
1 2 3
4 5 6 , Transpose (3x2):  1 4,2 5,3 6
```

Comprehensive Project Example

```java
// ==================== STUDENT CLASS ====================
class Student {
    // Encapsulated attributes
    private String studentId;
    private String name;
    private int age;
    private String department;
    private double[] marks;  // Array for subject marks
    private static int totalStudents = 0;
    private static final int MAX_SUBJECTS = 5;

    // Constructors
    public Student(String name, int age, String department) {
        this.studentId = "STU" + String.format("%04d", ++totalStudents)
            ;
        this.name = name;
        this.age = age;
        this.department = department;
        this.marks = new double[MAX_SUBJECTS];
        System.out.println("Student " + studentId + " created
            successfully.");
    }

    // Overloaded constructor
    public Student(String name, int age, String department, double[]
        marks) {
        this(name, age, department);
        if (marks.length <= MAX_SUBJECTS) {
            System.arraycopy(marks, 0, this.marks, 0, marks.length);
        }
    }

    // Getter methods
    public String getStudentId() { return studentId; }
    public String getName() { return name; }
    public int getAge() { return age; }
    public String getDepartment() { return department; }
    public static int getTotalStudents() { return totalStudents; }

    // Setter methods with validation
    public void setName(String name) {
        if (name != null && name.length() >= 2) {
            this.name = name;
        }
    }

    public void setAge(int age) {
        if (age >= 16 && age <= 60) {
            this.age = age;
        }
    }

    // Method to set marks for a specific subject
    public void setMark(int subjectIndex, double mark) {
        if (subjectIndex >= 0 && subjectIndex < MAX_SUBJECTS && mark >=
            0 && mark <= 100) {
            marks[subjectIndex] = mark;
        }
    }

    // Method to set all marks at once
    public void setAllMarks(double[] marks) {
        if (marks.length <= MAX_SUBJECTS) {
            System.arraycopy(marks, 0, this.marks, 0, marks.length);
        }
    }

    // Calculate total marks
    public double calculateTotal() {
        double total = 0;
        for (double mark : marks) {
            total += mark;
        }
        return total;
    }

    // Calculate average marks
    public double calculateAverage() {
        int count = 0;
        double sum = 0;
        for (double mark : marks) {
            if (mark > 0) {
                sum += mark;
                count++;
            }
        }
        return count > 0 ? sum / count : 0;
    }

    // Find maximum mark
    public double findMaxMark() {
        double max = marks[0];
        for (int i = 1; i < marks.length; i++) {
            if (marks[i] > max) {
                max = marks[i];
            }
        }
        return max;
    }

    // Find minimum mark
    public double findMinMark() {
        double min = marks[0];
        for (int i = 1; i < marks.length; i++) {
            if (marks[i] < min && marks[i] > 0) {
                min = marks[i];
            }
        }
        return min;
    }

    // Determine grade based on average
    public String calculateGrade() {
        double avg = calculateAverage();
        if (avg >= 90) return "A+";
        else if (avg >= 80) return "A";
        else if (avg >= 70) return "B";
        else if (avg >= 60) return "C";
        else if (avg >= 50) return "D";
        else return "F";
    }

    // Check if student passed
    public boolean isPassed() {
        return calculateAverage() >= 40;
    }

    // Display student information
    public void displayInfo() {
        System.out.println("\n" + "=".repeat(50));
        System.out.println("STUDENT ID: " + studentId);
        System.out.println("NAME: " + name);
        System.out.println("AGE: " + age);
        System.out.println("DEPARTMENT: " + department);
        System.out.println("-".repeat(50));

        System.out.println("MARKS:");
        String[] subjects = {"Math", "Physics", "Chemistry", "English", "Computer"};
        for (int i = 0; i < marks.length; i++) {
            if (marks[i] > 0) {
                System.out.printf("%-10s: %.2f\n", subjects[i], marks[i]);
            }
        }

        System.out.println("-".repeat(50));
        System.out.printf("TOTAL MARKS: %.2f\n", calculateTotal());
        System.out.printf("AVERAGE: %.2f\n", calculateAverage());
        System.out.printf("MAXIMUM: %.2f\n", findMaxMark());
        System.out.printf("MINIMUM: %.2f\n", findMinMark());
        System.out.println("GRADE: " + calculateGrade());
        System.out.println("STATUS: " + (isPassed() ? "PASSED" : "FAILED"));
        System.out.println("=".repeat(50));
    }

    // Static method to compare two students
    public static Student compareByMarks(Student s1, Student s2) {
        return s1.calculateAverage() >= s2.calculateAverage() ? s1 : s2;
    }
}

// ==================== MAIN CLASS ====================
public class StudentManagementSystem {
    public static void main(String[] args) {
        System.out.println("=== STUDENT MANAGEMENT SYSTEM ===\n");

        // Create array of students
        Student[] students = new Student[3];

        // Initialize students with different constructors
        students[0] = new Student("Aarav Sharma", 20, "Computer Science
            ");
        students[0].setAllMarks(new double[]{85.5, 90.0, 88.5, 92.0,
            87.5});

        students[1] = new Student("Priya Patel", 21, "Electronics");
        students[1].setMark(0, 78.0);
        students[1].setMark(1, 82.5);
        students[1].setMark(2, 75.0);
        students[1].setMark(3, 80.0);
        students[1].setMark(4, 85.5);

        students[2] = new Student("Rohan Singh", 19, "Mechanical",
                                  new double[]{65.0, 70.5, 68.0, 72.5,
                                      67.0});

        // Display all students
        System.out.println("\n=== ALL STUDENTS INFORMATION ===");
        for (Student student : students) {
            student.displayInfo();
        }

        // Perform operations
        System.out.println("\n=== PERFORMING OPERATIONS ===");

        // Update student information
        students[0].setName("Aarav Kumar Sharma");
        students[1].setAge(22);

        // Calculate class statistics
        System.out.println("\n=== CLASS STATISTICS ===");
        System.out.println("Total Students: " + Student.
            getTotalStudents());

        double classTotal = 0;
        Student topper = students[0];

        for (Student student : students) {
            classTotal += student.calculateAverage();
            if (student.calculateAverage() > topper.calculateAverage())
                {
                topper = student;
            }
        }

        double classAverage = classTotal / students.length;
        System.out.printf("Class Average: %.2f\n", classAverage);
        System.out.println("Class Topper: " + topper.getName() +
                         " (ID: " + topper.getStudentId() + ")");

        // Compare two students
        System.out.println("\n=== COMPARING STUDENTS ===");
        Student betterStudent = Student.compareByMarks(students[0],
            students[1]);
        System.out.println("Better performer between " + students[0].getName() +
                         " and " + students[1].getName() + ": " +
                         betterStudent.getName());

        // Find students who passed
        System.out.println("\n=== PASSED STUDENTS ===");
        for (Student student : students) {
            if (student.isPassed()) {
                System.out.println(student.getName() + " - PASSED with grade " +
                                 student.calculateGrade());
            }
        }

        // Department-wise analysis
        System.out.println("\n=== DEPARTMENT ANALYSIS ===");
        String[] departments = {"Computer Science", "Electronics", "Mechanical"};

        for (String dept : departments) {
            double deptTotal = 0;
            int deptCount = 0;

            for (Student student : students) {
                if (student.getDepartment().equals(dept)) {
                    deptTotal += student.calculateAverage();
                    deptCount++;
                }
            }

            if (deptCount > 0) {
                double deptAverage = deptTotal / deptCount;
                System.out.printf("%-20s: %.2f average (%d students)\n",
                                dept, deptAverage, deptCount);
            }
        }

        // Array operations on students
        System.out.println("\n=== STUDENT ARRAY OPERATIONS ===");

        // Create array of student names
        String[] studentNames = new String[students.length];
        for (int i = 0; i < students.length; i++) {
            studentNames[i] = students[i].getName();
        }

        System.out.println("Student Names: ");
        for (String name : studentNames) {
            System.out.println("- " + name);
        }

        // Create 2D array for marks overview
        double[][] marksOverview = new double[students.length][5];
        for (int i = 0; i < students.length; i++) {
            for (int j = 0; j < 5; j++) {
```

```java
                // In real scenario, we would have getter for individual marks
                marksOverview[i][j] = students[i].calculateAverage();
            }
        }

        System.out.println("\nMarks Overview Matrix:");
        System.out.print("Student\t\t");
        for (int i = 1; i <= 5; i++) System.out.print("Sub" + i + "\t");
        System.out.println();

        for (int i = 0; i < students.length; i++) {
            System.out.printf("%-15s", students[i].getName());
            for (int j = 0; j < 5; j++) {
                System.out.printf("%.1f\t", marksOverview[i][j]);
            }
            System.out.println();
        }
    }
}
```

Listing 12: Student Management System - Complete Implementation

StudentManagementSystem Program Output (Partial)

```
=== STUDENT MANAGEMENT SYSTEM ===
Student STU0001 created successfully.
Student STU0002 created successfully.
Student STU0003 created successfully.
=== ALL STUDENTS INFORMATION ===
==================================================
STUDENT ID: STU0001
NAME: Aarav Sharma
AGE: 20
DEPARTMENT: Computer Science
--------------------------------------------------
MARKS:
Math      : 85.50
Physics   : 90.00
Chemistry : 88.50
English   : 92.00
Computer  : 87.50
--------------------------------------------------
TOTAL MARKS: 443.50
AVERAGE: 88.70
MAXIMUM: 92.00
MINIMUM: 85.50
GRADE: A
STATUS: PASSED
==================================================
[Similar displays for other students...]
=== PERFORMING OPERATIONS ===
=== CLASS STATISTICS ===
Total Students: 3
Class Average: 78.57
Class Topper: Aarav Kumar Sharma (ID: STU0001)
=== COMPARING STUDENTS ===
Better performer between Aarav Kumar Sharma and Priya Patel: Aarav Kumar Sharma
=== PASSED STUDENTS ===
Aarav Kumar Sharma - PASSED with grade A
Priya Patel - PASSED with grade B
Rohan Singh - PASSED with grade C
=== DEPARTMENT ANALYSIS ===
Computer Science   : 88.70 average (1 students)
Electronics        : 80.20 average (1 students)
Mechanical         : 68.60 average (1 students)
=== STUDENT ARRAY OPERATIONS ===
Student Names:
- Aarav Kumar Sharma
- Priya Patel
- Rohan Singh
Marks Overview Matrix:
Student  Sub1 Sub2 Sub3 Sub4 Sub5
Aarav Kumar Sharma 88.7 88.7 88.7 88.7 88.7
Priya Patel 80.2 80.2 80.2 80.2 80.2
Rohan Singh 68.6 68.6 68.6 68.6 68.6
```

Unit Summary and Learning Outcomes

Key Concepts Covered

1. **Java Fundamentals:** History, JVM architecture, compilation process
2. **Basic Syntax:** Variables, data types, operators, type casting
3. **Control Structures:** If-else, switch, loops with practical patterns
4. **OOP Principles:** Classes, objects, encapsulation, access modifiers
5. **Methods:** Overloading, constructors, 'this' keyword, static methods
6. **Arrays:** Single/multi-dimensional arrays, matrix operations, practical applications

Practical Skills Developed

- Setting up Java development environment
- Writing, compiling, and executing Java programs
- Implementing real-world systems using OOP concepts
- Debugging and testing Java applications
- Using arrays for data organization and processing
- Applying encapsulation for data security

Assessment Methods

Component	**Weightage**
Programming Assignments	30%
Laboratory Exercises	20%
Mid-Term Examination	20%
End-Term Examination	30%

Recommended Practice Problems

1. Create a `Book` class with encapsulation and implement a library system
2. Develop a calculator that uses method overloading for different operations
3. Implement a matrix class with methods for addition, multiplication, and transpose
4. Create a banking system with multiple account types using inheritance
5. Design a student grade management system using 2D arrays
6. Implement a shopping cart system with array of objects

References and Further Reading

1. Oracle Java Documentation: `https://docs.oracle.com/javase/`
2. Java Tutorials: `https://docs.oracle.com/javase/tutorial/`
3. Java Programming Exercises: `https://www.w3resource.com/java-exercises/`
4. Codecademy Java Course: `https://www.codecademy.com/learn/learn-java`
5. GeeksforGeeks Java Programming: `https://www.geeksforgeeks.org/java/`

End of Unit I Syllabus