Fundamentals of Java Programming - 4 Hour Course

📚 Course Overview

Duration: 4 hours
Target Audience: Beginners with basic Java introduction knowledge
Prerequisites: Understanding of JVM, bytecode, and basic Java concepts
Objectives: Master core Java programming constructs and OOP fundamentals

---

⏰ Course Schedule & Session Breakdown

Session 1: Object-Oriented Concepts & Java Basics (60 minutes)

- Object-Oriented Programming Introduction (20 min)
- Java Keywords & Reserved Words (15 min)
- Data Types & Variables (25 min)

Session 2: Operators & Type Casting (60 minutes)

- Java Operators (35 min)
- Type Casting & Conversion (25 min)

Session 3: Control Flow & Decision Making (60 minutes)

- Conditional Statements (if, else, switch) (30 min)
- Ternary Operator (10 min)
- Control Statements (loops) (20 min)

Session 4: Advanced Control & Arrays (60 minutes)

- Advanced Loop Concepts (20 min)
- Static Keyword (15 min)
- Arrays Fundamentals (25 min)

🎯 SESSION 1: OBJECT-ORIENTED CONCEPTS & JAVA BASICS

1.1 Object-Oriented Programming (OOP) Concepts (20 minutes)

🏛️ What is Object-Oriented Programming?

OOP is a programming paradigm that organizes code around objects and classes rather than functions and logic.

🎭 The Four Pillars of OOP

1. 📦 Encapsulation - Data Hiding

```
public class BankAccount {
    private double balance;        // Private data - hidden from outside
    private String accountNumber;

    // Public methods - controlled access
    public void deposit(double amount) {
        if (amount > 0) {
            balance += amount;
            System.out.println("Deposited: $" + amount);
        }
    }

    public double getBalance() {   // Getter method
        return balance;
    }

    public void setAccountNumber(String accNum) {  // Setter method
        if (accNum != null && accNum.length() >= 8) {
            this.accountNumber = accNum;
        }
    }
}

// Usage Example
public class BankDemo {
    public static void main(String[] args) {
        BankAccount account = new BankAccount();

        // This works - using public methods
        account.deposit(1000);
        System.out.println("Balance: $" + account.getBalance());

        // This won't work - balance is private
        // account.balance = 5000;  // ❌ Compilation error!
    }
}
```

2. 🚀 Inheritance - Code Reusability

```
// Parent class (Superclass)
public class Vehicle {
    protected String brand;
    protected int year;
    protected double speed;

    public Vehicle(String brand, int year) {
        this.brand = brand;
        this.year = year;
```

```
        this.speed = 0;
    }

    public void start() {
        System.out.println(brand + " is starting...");
    }

    public void accelerate(double increment) {
        speed += increment;
        System.out.println("Speed: " + speed + " mph");
    }
}

// Child class (Subclass)
public class Car extends Vehicle {
    private int numberOfDoors;

    public Car(String brand, int year, int doors) {
        super(brand, year);  // Call parent constructor
        this.numberOfDoors = doors;
    }

    // Method specific to Car
    public void honk() {
        System.out.println("Beep beep! " + brand + " is honking!");
    }

    // Override parent method
    @Override
    public void start() {
        System.out.println("Starting " + brand + " car with key ignition");
    }
}

// Another child class
public class Motorcycle extends Vehicle {
    private boolean hasSidecar;

    public Motorcycle(String brand, int year, boolean sidecar) {
        super(brand, year);
        this.hasSidecar = sidecar;
    }

    public void wheelie() {
        System.out.println(brand + " motorcycle doing a wheelie!");
    }

    @Override
    public void start() {
        System.out.println("Kick-starting " + brand + " motorcycle");
    }
}

// Demonstration
```

```
public class VehicleDemo {
    public static void main(String[] args) {
        Car myCar = new Car("Toyota", 2023, 4);
        Motorcycle myBike = new Motorcycle("Harley", 2022, false);

        // Both inherit from Vehicle
        myCar.start();           // Car's version
        myCar.accelerate(30);    // Inherited method
        myCar.honk();            // Car-specific method

        myBike.start();          // Motorcycle's version
        myBike.accelerate(50);   // Inherited method
        myBike.wheelie();        // Motorcycle-specific method
    }
}
```

3. 🎭 Polymorphism - Many Forms

```
public class PolymorphismDemo {
    public static void main(String[] args) {
        // Same reference type, different object types
        Vehicle vehicle1 = new Car("Honda", 2023, 4);
        Vehicle vehicle2 = new Motorcycle("Yamaha", 2022, false);

        // Polymorphic method calls
        startVehicle(vehicle1);  // Calls Car's start() method
        startVehicle(vehicle2);  // Calls Motorcycle's start() method

        // Array of different vehicle types
        Vehicle[] garage = {
            new Car("BMW", 2023, 2),
            new Motorcycle("Ducati", 2022, false),
            new Car("Tesla", 2024, 4)
        };

        // Polymorphic iteration
        for (Vehicle v : garage) {
            v.start();           // Different start() for each type
            v.accelerate(25);    // Same method for all
        }
    }

    // Method that works with any Vehicle type
    public static void startVehicle(Vehicle v) {
        System.out.println("Starting vehicle...");
        v.start();  // Polymorphic call - actual method depends on object type
    }
}
```

4. 🎨 Abstraction - Hiding Complexity

```java
// Abstract class - cannot be instantiated
abstract class Shape {
    protected String color;

    public Shape(String color) {
        this.color = color;
    }

    // Abstract method - must be implemented by subclasses
    public abstract double calculateArea();
    public abstract double calculatePerimeter();

    // Concrete method - inherited by all subclasses
    public void displayInfo() {
        System.out.println("This is a " + color + " shape");
        System.out.println("Area: " + calculateArea());
        System.out.println("Perimeter: " + calculatePerimeter());
    }
}

class Circle extends Shape {
    private double radius;

    public Circle(String color, double radius) {
        super(color);
        this.radius = radius;
    }

    @Override
    public double calculateArea() {
        return Math.PI * radius * radius;
    }

    @Override
    public double calculatePerimeter() {
        return 2 * Math.PI * radius;
    }
}

class Rectangle extends Shape {
    private double length, width;

    public Rectangle(String color, double length, double width) {
        super(color);
        this.length = length;
        this.width = width;
    }

    @Override
    public double calculateArea() {
        return length * width;
    }
```

```
    @Override
    public double calculatePerimeter() {
        return 2 * (length + width);
    }
}

// Usage
public class ShapeDemo {
    public static void main(String[] args) {
        // Shape shape = new Shape("red");  // ❌ Cannot instantiate abstract
class

        Shape circle = new Circle("Red", 5.0);
        Shape rectangle = new Rectangle("Blue", 4.0, 6.0);

        circle.displayInfo();
        System.out.println();
        rectangle.displayInfo();
    }
}
```

🎯 Key OOP Benefits

- **Code Reusability:** Write once, use many times through inheritance
- **Modularity:** Break complex problems into smaller, manageable objects
- **Security:** Encapsulation protects data from unauthorized access
- **Flexibility:** Polymorphism allows objects to be treated uniformly
- **Maintainability:** Changes to one class don't affect others (when properly designed)

1.2 Java Keywords & Reserved Words (15 minutes)

🔑 Java Keywords Overview

Java has 50 reserved keywords that have special meaning and cannot be used as identifiers.

📋 Complete Keywords List

Access Modifiers

```
public class AccessDemo {
    public int publicVar = 1;        // Accessible everywhere
    private int privateVar = 2;      // Only within this class
    protected int protectedVar = 3;  // Package + subclasses
    int packageVar = 4;              // Package-private (default)
}
```

Class, Method, and Variable Modifiers

```
// Class modifiers
public final class FinalClass { }           // Cannot be extended
abstract class AbstractClass { }            // Cannot be instantiated

public class ModifierDemo {
    // Variable modifiers
    static int staticVar = 10;               // Belongs to class, not instance
    final int FINAL_VAR = 20;               // Cannot be changed
    volatile int volatileVar = 30;           // Thread-safe visibility
    transient int transientVar = 40;        // Not serialized

    // Method modifiers
    static void staticMethod() { }           // Called on class, not instance
    final void finalMethod() { }             // Cannot be overridden
    synchronized void syncMethod() { }       // Thread-safe
    native void nativeMethod();             // Implemented in native code
}
```

Control Flow Keywords

```
public class ControlFlowDemo {
    public static void main(String[] args) {
        // Conditional keywords
        if (true) {
            System.out.println("If statement");
        } else {
            System.out.println("Else statement");
        }

        // Switch statement
        int day = 3;
        switch (day) {
            case 1:
                System.out.println("Monday");
                break;
            case 2:
                System.out.println("Tuesday");
                break;
            default:
                System.out.println("Other day");
        }

        // Loop keywords
        for (int i = 0; i < 3; i++) {
            if (i == 1) continue;    // Skip this iteration
            if (i == 2) break;       // Exit loop
            System.out.println("For loop: " + i);
        }

        int count = 0;
```

```
        while (count < 2) {
            System.out.println("While loop: " + count);
            count++;
        }

        do {
            System.out.println("Do-while executes at least once");
        } while (false);
    }
}
```

Exception Handling Keywords

```
public class ExceptionDemo {
    public static void main(String[] args) {
        try {
            int result = 10 / 0;  // This will throw an exception
        } catch (ArithmeticException e) {
            System.out.println("Caught exception: " + e.getMessage());
        } finally {
            System.out.println("Finally block always executes");
        }
    }

    // Method that throws an exception
    public static void riskyMethod() throws Exception {
        throw new Exception("Something went wrong!");
    }
}
```

Object-Oriented Keywords

```
// Class definition
class Parent {
    protected String name = "Parent";

    public void greet() {
        System.out.println("Hello from Parent");
    }
}

class Child extends Parent {    // Inheritance
    public Child() {
        super();                // Call parent constructor
        this.name = "Child";    // Refer to current object
    }

    @Override
```

```
    public void greet() {
        super.greet();          // Call parent method
        System.out.println("Hello from Child");
    }

    public void demonstrateThis() {
        String name = "Local";
        System.out.println("Local variable: " + name);
        System.out.println("Instance variable: " + this.name);
    }
}

// Interface definition
interface Drawable {
    void draw();                // Abstract method (implicitly public abstract)
}

class Circle implements Drawable {
    public void draw() {
        System.out.println("Drawing a circle");
    }
}
```

Package and Import Keywords

```
package com.example.myapp;      // Package declaration

import java.util.List;          // Import specific class
import java.util.*;             // Import all classes from package

public class PackageDemo {
    // Class implementation
}
```

Primitive Type Keywords

```
public class PrimitiveDemo {
    // Primitive data types
    boolean flag = true;          // true or false
    byte smallNumber = 127;       // -128 to 127
    short mediumNumber = 32000;   // -32,768 to 32,767
    int number = 100000;          // -2^31 to 2^31-1
    long bigNumber = 100000L;     // -2^63 to 2^63-1
    float decimal = 3.14f;        // Single precision
    double precision = 3.14159; // Double precision
    char character = 'A';         // Unicode character

    // Special values
```

```
    void demonstrateNull() {
        String text = null;      // null reference
        if (text instanceof String) {  // instanceof operator
            System.out.println("text is a String");
        }
    }
}
```

🚫 Reserved Keywords (Not Currently Used)

```
// These are reserved for future use:
// const - use 'final' instead
// goto - not implemented in Java (considered harmful)
```

🎯 Important Notes About Keywords

- **Case-sensitive:** `Public` is not the same as `public`
- **Cannot be used as identifiers:** Variable names, method names, class names
- **Context-sensitive:** Some have different meanings in different contexts

❌ Common Mistakes

```
// These will cause compilation errors:
// int class = 5;           // 'class' is a keyword
// String if = "test";      // 'if' is a keyword
// boolean true = false;    // 'true' is a keyword

// These are valid (not keywords):
int Class = 5;              // ✅ Capital 'C'
String If = "test";         // ✅ Capital 'I'
boolean TRUE = false;       // ✅ All capitals
```

1.3 Data Types & Variables (25 minutes)

🎯 Java Data Types Overview

Java has two main categories of data types:

1. **Primitive Data Types** - Store actual values
2. **Reference Data Types** - Store references to objects

🏗️ Primitive Data Types

📊 Numeric Data Types

Integer Types:

```
public class IntegerTypes {
    public static void main(String[] args) {
        // byte: 8-bit signed integer (-128 to 127)
        byte temperature = 25;
        byte freezing = -10;
        System.out.println("Temperature: " + temperature + "°C");

        // short: 16-bit signed integer (-32,768 to 32,767)
        short population = 15000;
        short deficit = -5000;
        System.out.println("City population: " + population);

        // int: 32-bit signed integer (-2^31 to 2^31-1)
        int distance = 384400;  // Distance to moon in km
        int debt = -50000;
        System.out.println("Distance to moon: " + distance + " km");

        // long: 64-bit signed integer (-2^63 to 2^63-1)
        long worldPopulation = 8000000000L;  // Note the 'L' suffix
        long lightYear = 9460730472580800L;
        System.out.println("World population: " + worldPopulation);

        // Size demonstration
        System.out.println("\nInteger Type Ranges:");
        System.out.println("byte: " + Byte.MIN_VALUE + " to " + Byte.MAX_VALUE);
        System.out.println("short: " + Short.MIN_VALUE + " to " +
Short.MAX_VALUE);
        System.out.println("int: " + Integer.MIN_VALUE + " to " +
Integer.MAX_VALUE);
        System.out.println("long: " + Long.MIN_VALUE + " to " + Long.MAX_VALUE);
    }
}
```

Floating-Point Types:

```
public class FloatingTypes {
    public static void main(String[] args) {
        // float: 32-bit IEEE 754 floating point
        float price = 19.99f;           // Note the 'f' suffix
        float pi = 3.14159f;
        float scientific = 1.23e-4f;    // Scientific notation

        // double: 64-bit IEEE 754 floating point (default for decimals)
        double precise = 3.141592653589793;
        double large = 1.7976931348623157e+308;
        double small = 4.9e-324;

        System.out.println("Float precision: " + price);
        System.out.println("Double precision: " + precise);
```

```
        // Precision comparison
        float f = 0.1f + 0.2f;
        double d = 0.1 + 0.2;
        System.out.println("\nPrecision differences:");
        System.out.println("float (0.1 + 0.2): " + f);
        System.out.println("double (0.1 + 0.2): " + d);

        // Special values
        System.out.println("\nSpecial floating-point values:");
        System.out.println("Positive infinity: " + Double.POSITIVE_INFINITY);
        System.out.println("Negative infinity: " + Double.NEGATIVE_INFINITY);
        System.out.println("Not a Number: " + Double.NaN);
        System.out.println("Max double: " + Double.MAX_VALUE);
        System.out.println("Min double: " + Double.MIN_VALUE);
    }
}
```

☑ Boolean Type

```
public class BooleanDemo {
    public static void main(String[] args) {
        // boolean: true or false only
        boolean isJavaFun = true;
        boolean isRaining = false;
        boolean testResult = (5 > 3);  // Expression result

        System.out.println("Is Java fun? " + isJavaFun);
        System.out.println("Is it raining? " + isRaining);
        System.out.println("Is 5 > 3? " + testResult);

        // Boolean operations
        boolean and = true && false;    // Logical AND
        boolean or = true || false;     // Logical OR
        boolean not = !true;            // Logical NOT

        System.out.println("\nBoolean operations:");
        System.out.println("true && false = " + and);
        System.out.println("true || false = " + or);
        System.out.println("!true = " + not);

        // Conditional usage
        if (isJavaFun) {
            System.out.println("Let's continue learning Java!");
        }
    }
}
```

🔤 Character Type

```
public class CharacterDemo {
    public static void main(String[] args) {
        // char: 16-bit Unicode character
        char letter = 'A';
        char digit = '5';
        char symbol = '@';
        char space = ' ';

        // Unicode literals
        char heart = '\u2764';          // ♡
        char smiley = '\u263A';         // ☺
        char alpha = '\u03B1';          // α

        // Escape sequences
        char newline = '\n';
        char tab = '\t';
        char backslash = '\\';
        char quote = '\'';

        System.out.println("Basic characters:");
        System.out.println("Letter: " + letter);
        System.out.println("Digit: " + digit);
        System.out.println("Symbol: " + symbol);

        System.out.println("\nUnicode characters:");
        System.out.println("Heart: " + heart);
        System.out.println("Smiley: " + smiley);
        System.out.println("Alpha: " + alpha);

        System.out.println("\nEscape sequences:");
        System.out.println("Line 1" + newline + "Line 2");
        System.out.println("Column1" + tab + "Column2");
        System.out.println("Backslash: " + backslash);
        System.out.println("Quote: " + quote);

        // Character arithmetic
        char a = 'A';
        char b = (char)(a + 1);  // Casting required
        System.out.println("\nCharacter arithmetic:");
        System.out.println("'A' + 'B' as int: " + charSum);
        System.out.println("'A' + 'B' as char: '" + charSumCast + "'");

        // 6. Wrapper class conversions
        Integer wrapperInt = 100;           // Autoboxing: int → Integer
        int primitiveInt = wrapperInt;      // Unboxing: Integer → int

        // Null pointer exception risk
        Integer nullInteger = null;
        try {
            int dangerous = nullInteger;     // NullPointerException!
        } catch (NullPointerException e) {
            System.out.println("Null unboxing error: " +
e.getClass().getSimpleName());
```

```
        }

        // Best practices summary
        System.out.println("\n=== BEST PRACTICES ===");
        System.out.println("1. Always check ranges before narrowing casts");
        System.out.println("2. Use explicit casting for clarity");
        System.out.println("3. Handle parsing exceptions");
        System.out.println("4. Be aware of integer division vs floating-point
division");
        System.out.println("5. Watch out for null values in wrapper classes");
    }
}
```

🔧 Wrapper Classes and Boxing/Unboxing

```
public class WrapperClasses {
    public static void main(String[] args) {
        System.out.println("=== WRAPPER CLASSES ===");

        // Primitive types and their wrapper classes
        byte primitiveByte = 10;
        Byte wrapperByte = primitiveByte;           // Autoboxing

        short primitiveShort = 100;
        Short wrapperShort = Short.valueOf(primitiveShort);  // Explicit boxing

        int primitiveInt = 1000;
        Integer wrapperInt = primitiveInt;          // Autoboxing

        long primitiveLong = 10000L;
        Long wrapperLong = primitiveLong;           // Autoboxing

        float primitiveFloat = 3.14f;
        Float wrapperFloat = primitiveFloat;        // Autoboxing

        double primitiveDouble = 2.718;
        Double wrapperDouble = primitiveDouble;     // Autoboxing

        boolean primitiveBoolean = true;
        Boolean wrapperBoolean = primitiveBoolean;  // Autoboxing

        char primitiveChar = 'X';
        Character wrapperChar = primitiveChar;      // Autoboxing

        System.out.println("Autoboxing examples:");
        System.out.println("int " + primitiveInt + " → Integer " + wrapperInt);
        System.out.println("double " + primitiveDouble + " → Double " +
wrapperDouble);
        System.out.println("boolean " + primitiveBoolean + " → Boolean " +
wrapperBoolean);
```

```
        // Unboxing (wrapper to primitive)
        int unboxedInt = wrapperInt;                // Auto-unboxing
        double unboxedDouble = wrapperDouble.doubleValue();  // Explicit unboxing
        boolean unboxedBoolean = wrapperBoolean;    // Auto-unboxing

        System.out.println("\nUnboxing examples:");
        System.out.println("Integer " + wrapperInt + " → int " + unboxedInt);
        System.out.println("Double " + wrapperDouble + " → double " +
unboxedDouble);
        System.out.println("Boolean " + wrapperBoolean + " → boolean " +
unboxedBoolean);

        // Wrapper class utility methods
        System.out.println("\nWrapper class utility methods:");

        // Parsing strings
        String numberString = "12345";
        int parsed = Integer.parseInt(numberString);
        System.out.println("Parsed \"" + numberString + "\" to int: " + parsed);

        // Converting to different bases
        String binary = Integer.toBinaryString(15);
        String octal = Integer.toOctalString(15);
        String hex = Integer.toHexString(15);
        System.out.println("15 in binary: " + binary);
        System.out.println("15 in octal: " + octal);
        System.out.println("15 in hex: " + hex);

        // Min/Max values
        System.out.println("Integer.MAX_VALUE: " + Integer.MAX_VALUE);
        System.out.println("Integer.MIN_VALUE: " + Integer.MIN_VALUE);
        System.out.println("Double.MAX_VALUE: " + Double.MAX_VALUE);
        System.out.println("Double.MIN_VALUE: " + Double.MIN_VALUE);

        // Comparison
        Integer int1 = 100;
        Integer int2 = 100;
        Integer int3 = new Integer(100);

        System.out.println("\nWrapper class comparison:");
        System.out.println("int1 == int2: " + (int1 == int2));         // true
(cache)
        System.out.println("int1 == int3: " + (int1 == int3));         // false
(different objects)
        System.out.println("int1.equals(int3): " + int1.equals(int3)); // true
(same value)

        // Integer cache (-128 to 127)
        Integer cached1 = 127;
        Integer cached2 = 127;
        Integer notCached1 = 128;
        Integer notCached2 = 128;

        System.out.println("\nInteger caching:");
```

```
        System.out.println("127 == 127: " + (cached1 == cached2));          // true
(cached)
        System.out.println("128 == 128: " + (notCached1 == notCached2));    //
false (not cached)
    }
}
```

🎯 SESSION 3: CONTROL FLOW & DECISION MAKING

3.1 Conditional Statements (30 minutes)

🔥 If Statements

```
public class IfStatements {
    public static void main(String[] args) {
        System.out.println("=== IF STATEMENTS ===");

        // Simple if statement
        int temperature = 25;

        if (temperature > 20) {
            System.out.println("It's warm outside!");
        }

        // If-else statement
        int age = 17;

        if (age >= 18) {
            System.out.println("You can vote!");
        } else {
            System.out.println("You cannot vote yet.");
        }

        // If-else if-else ladder
        int score = 85;

        if (score >= 90) {
            System.out.println("Grade: A (Excellent!)");
        } else if (score >= 80) {
            System.out.println("Grade: B (Good!)");
        } else if (score >= 70) {
            System.out.println("Grade: C (Average)");
        } else if (score >= 60) {
            System.out.println("Grade: D (Below Average)");
        } else {
            System.out.println("Grade: F (Fail)");
        }

        // Nested if statements
```

```
boolean hasLicense = true;
int driverAge = 20;
boolean hasInsurance = false;

if (hasLicense) {
    if (driverAge >= 18) {
        if (hasInsurance) {
            System.out.println("You can drive legally!");
        } else {
            System.out.println("You need insurance to drive.");
        }
    } else {
        System.out.println("You must be 18 or older to drive.");
    }
} else {
    System.out.println("You need a driver's license.");
}

// Complex conditions with logical operators
boolean isWeekend = true;
boolean isHoliday = false;
boolean hasWork = false;

if ((isWeekend || isHoliday) && !hasWork) {
    System.out.println("Time to relax!");
} else if (hasWork && (isWeekend || isHoliday)) {
    System.out.println("Working on a day off :(");
} else {
    System.out.println("Regular workday.");
}

// If statement without braces (single statement)
int number = 15;

if (number % 2 == 0)
    System.out.println(number + " is even");
else
    System.out.println(number + " is odd");

// Best practice: always use braces for clarity
if (number > 10) {
    System.out.println("Number is greater than 10");
    System.out.println("This makes the code more readable");
}

// Common pitfalls
int x = 5;

// Pitfall 1: Assignment instead of comparison
// if (x = 10) {  // ✗ This would be assignment, not comparison
//     System.out.println("This won't compile");
// }

// Correct comparison
```

```
        if (x == 10) {
            System.out.println("x equals 10");
        }

        // Pitfall 2: Dangling else
        if (x > 0)
            if (x < 10)
                System.out.println("x is between 0 and 10");
        else  // This else belongs to the inner if, not the outer if!
            System.out.println("x is not less than 10");

        // Better with braces
        if (x > 0) {
            if (x < 10) {
                System.out.println("x is between 0 and 10");
            } else {
                System.out.println("x is 10 or greater");
            }
        }
    }
}
```

Switch Statements

```
public class SwitchStatements {
    public static void main(String[] args) {
        System.out.println("=== SWITCH STATEMENTS ===");

        // Basic switch statement
        int dayOfWeek = 3;

        switch (dayOfWeek) {
            case 1:
                System.out.println("Monday - Start of work week");
                break;
            case 2:
                System.out.println("Tuesday - Getting into the groove");
                break;
            case 3:
                System.out.println("Wednesday - Hump day!");
                break;
            case 4:
                System.out.println("Thursday - Almost there");
                break;
            case 5:
                System.out.println("Friday - TGIF!");
                break;
            case 6:
                System.out.println("Saturday - Weekend fun");
                break;
            case 7:
```

```
                System.out.println("Sunday - Rest day");
                break;
            default:
                System.out.println("Invalid day");
        }

        // Switch with char
        char grade = 'B';

        switch (grade) {
            case 'A':
                System.out.println("Excellent work!");
                break;
            case 'B':
                System.out.println("Good job!");
                break;
            case 'C':
                System.out.println("Average performance");
                break;
            case 'D':
                System.out.println("Below average");
                break;
            case 'F':
                System.out.println("Failed");
                break;
            default:
                System.out.println("Invalid grade");
        }

        // Switch with String (Java 7+)
        String month = "December";

        switch (month) {
            case "December":
            case "January":
            case "February":
                System.out.println(month + " is a winter month");
                break;
            case "March":
            case "April":
            case "May":
                System.out.println(month + " is a spring month");
                break;
            case "June":
            case "July":
            case "August":
                System.out.println(month + " is a summer month");
                break;
            case "September":
            case "October":
            case "November":
                System.out.println(month + " is a fall month");
                break;
            default:
```

```
            System.out.println("Invalid month");
        }

        // Fall-through behavior (no break)
        int number = 2;
        System.out.println("\nFall-through example for number " + number + ":");

        switch (number) {
            case 1:
                System.out.println("One");
                // Fall through to case 2
            case 2:
                System.out.println("Two");
                // Fall through to case 3
            case 3:
                System.out.println("Three");
                break;  // Stop here
            case 4:
                System.out.println("Four");
                break;
            default:
                System.out.println("Other number");
        }

        // Practical example: Calculator
        char operator = '+';
        double num1 = 10.5;
        double num2 = 3.2;
        double result = 0;

        switch (operator) {
            case '+':
                result = num1 + num2;
                System.out.println(num1 + " + " + num2 + " = " + result);
                break;
            case '-':
                result = num1 - num2;
                System.out.println(num1 + " - " + num2 + " = " + result);
                break;
            case '*':
                result = num1 * num2;
                System.out.println(num1 + " * " + num2 + " = " + result);
                break;
            case '/':
                if (num2 != 0) {
                    result = num1 / num2;
                    System.out.println(num1 + " / " + num2 + " = " + result);
                } else {
                    System.out.println("Error: Division by zero!");
                }
                break;
            default:
                System.out.println("Invalid operator: " + operator);
        }
```

```
        // Switch expressions (Java 14+) - Preview feature
        // Note: This requires Java 14+ with preview features enabled
        /*
        String dayType = switch (dayOfWeek) {
            case 1, 2, 3, 4, 5 -> "Weekday";
            case 6, 7 -> "Weekend";
            default -> "Invalid";
        };
        System.out.println("Day type: " + dayType);
        */

        // Equivalent using traditional switch
        String dayType;
        switch (dayOfWeek) {
            case 1:
            case 2:
            case 3:
            case 4:
            case 5:
                dayType = "Weekday";
                break;
            case 6:
            case 7:
                dayType = "Weekend";
                break;
            default:
                dayType = "Invalid";
        }
        System.out.println("Day type: " + dayType);

        // When to use switch vs if-else
        System.out.println("\n=== WHEN TO USE SWITCH VS IF-ELSE ===");
        System.out.println("Use SWITCH when:");
        System.out.println("- Testing a single variable against multiple exact
values");
        System.out.println("- Values are constants (literals)");
        System.out.println("- You have many discrete cases");
        System.out.println("- Working with enums");
        System.out.println();
        System.out.println("Use IF-ELSE when:");
        System.out.println("- Testing complex conditions");
        System.out.println("- Using ranges or relational operators");
        System.out.println("- Combining multiple variables");
        System.out.println("- Need boolean expressions");
    }
}
```

🔍 Advanced Conditional Examples

```
public class AdvancedConditionals {
    public static void main(String[] args) {
        System.out.println("=== ADVANCED CONDITIONAL EXAMPLES ===");

        // Example 1: Login System
        String username = "admin";
        String password = "secret123";
        boolean isActive = true;
        int loginAttempts = 2;

        System.out.println("=== LOGIN SYSTEM ===");
        if (username.equals("admin") && password.equals("secret123")) {
            if (isActive) {
                if (loginAttempts < 3) {
                    System.out.println("Login successful! Welcome, " + username);
                } else {
                    System.out.println("Account locked due to too many failed
attempts");
                }
            } else {
                System.out.println("Account is deactivated");
            }
        } else {
            System.out.println("Invalid username or password");
        }

        // Example 2: Shipping Calculator
        double orderAmount = 75.50;
        String customerType = "premium";
        String destination = "domestic";

        System.out.println("\n=== SHIPPING CALCULATOR ===");
        double shippingCost = 0;

        if (orderAmount >= 100) {
            shippingCost = 0;  // Free shipping for orders over $100
            System.out.println("Free shipping applied!");
        } else {
            switch (destination.toLowerCase()) {
                case "domestic":
                    shippingCost = customerType.equals("premium") ? 5.99 : 9.99;
                    break;
                case "international":
                    shippingCost = customerType.equals("premium") ? 15.99 : 25.99;
                    break;
                default:
                    shippingCost = 9.99;
            }
        }

        System.out.println("Order amount: $" + orderAmount);
        System.out.println("Customer type: " + customerType);
        System.out.println("Destination: " + destination);
```

```
        System.out.println("Shipping cost: $" + shippingCost);
        System.out.println("Total: $" + (orderAmount + shippingCost));

        // Example 3: Student Grade Calculator with Multiple Criteria
        int mathScore = 85;
        int scienceScore = 92;
        int englishScore = 78;
        int attendancePercent = 95;
        boolean hasExtraCurricular = true;

        System.out.println("\n=== STUDENT EVALUATION ===");

        double average = (mathScore + scienceScore + englishScore) / 3.0;
        char letterGrade;
        boolean isHonorRoll = false;
        String comments = "";

        // Determine letter grade
        if (average >= 90) {
            letterGrade = 'A';
        } else if (average >= 80) {
            letterGrade = 'B';
        } else if (average >= 70) {
            letterGrade = 'C';
        } else if (average >= 60) {
            letterGrade = 'D';
        } else {
            letterGrade = 'F';
        }

        // Honor roll criteria
        if (average >= 85 && attendancePercent >= 90 &&
            mathScore >= 80 && scienceScore >= 80 && englishScore >= 80) {
            isHonorRoll = true;
            comments += "Congratulations on making the Honor Roll! ";
        }

        // Additional comments based on performance
        if (mathScore < 70 || scienceScore < 70 || englishScore < 70) {
            comments += "Consider tutoring in weak subjects. ";
        }

        if (attendancePercent < 80) {
            comments += "Attendance needs improvement. ";
        }

        if (hasExtraCurricular) {
            comments += "Great job participating in extracurricular activities! ";
        }

        System.out.println("Math: " + mathScore);
        System.out.println("Science: " + scienceScore);
        System.out.println("English: " + englishScore);
        System.out.println("Average: " + String.format("%.1f", average));
```

```
        System.out.println("Letter Grade: " + letterGrade);
        System.out.println("Attendance: " + attendancePercent + "%");
        System.out.println("Honor Roll: " + (isHonorRoll ? "Yes" : "No"));
        System.out.println("Comments: " + comments);

        // Example 4: Date Validation
        int year = 2024;
        int month = 2;  // February
        int day = 29;

        System.out.println("\n=== DATE VALIDATION ===");
        boolean isValidDate = false;

        if (year > 0 && month >= 1 && month <= 12 && day >= 1) {
            int maxDays;

            switch (month) {
                case 1: case 3: case 5: case 7: case 8: case 10: case 12:
                    maxDays = 31;
                    break;
                case 4: case 6: case 9: case 11:
                    maxDays = 30;
                    break;
                case 2:
                    // Leap year calculation
                    boolean isLeapYear = (year % 4 == 0 && year % 100 != 0) ||
(year % 400 == 0);
                    maxDays = isLeapYear ? 29 : 28;
                    break;
                default:
                    maxDays = 0;
            }

            if (day <= maxDays) {
                isValidDate = true;
            }
        }

        System.out.println("Date: " + month + "/" + day + "/" + year);
        System.out.println("Is valid: " + isValidDate);

        if (!isValidDate) {
            System.out.println("Invalid date!");
        } else {
            // Determine day of week (simplified algorithm)
            String[] months = {"", "January", "February", "March", "April", "May",
"June",
                               "July", "August", "September", "October", "November",
"December"};
            System.out.println("Date: " + months[month] + " " + day + ", " +
year);
        }
    }
}
```

3.2 Ternary Operator (10 minutes)

```
public class TernaryOperator {
    public static void main(String[] args) {
        System.out.println("=== TERNARY OPERATOR ===");

        // Basic syntax: condition ? valueIfTrue : valueIfFalse
        int a = 15, b = 25;

        int max = (a > b) ? a : b;
        int min = (a < b) ? a : b;

        System.out.println("a = " + a + ", b = " + b);
        System.out.println("Maximum: " + max);
        System.out.println("Minimum: " + min);

        // String results
        String comparison = (a > b) ? "a is greater" : "b is greater or equal";
        System.out.println(comparison);

        // Ternary vs if-else comparison
        System.out.println("\n=== TERNARY VS IF-ELSE ===");

        int score = 85;

        // Using ternary operator
        String grade = (score >= 90) ? "A" :
                       (score >= 80) ? "B" :
                       (score >= 70) ? "C" :
                       (score >= 60) ? "D" : "F";

        // Equivalent if-else
        String gradeIfElse;
        if (score >= 90) {
            gradeIfElse = "A";
        } else if (score >= 80) {
            gradeIfElse = "B";
        } else if (score >= 70) {
            gradeIfElse = "C";
        } else if (score >= 60) {
            gradeIfElse = "D";
        } else {
            gradeIfElse = "F";
        }

        System.out.println("Score: " + score);
        System.out.println("Grade (ternary): " + grade);
        System.out.println("Grade (if-else): " + gradeIfElse);
```

```
        // Practical examples
        System.out.println("\n=== PRACTICAL EXAMPLES ===");

        // Example 1: Voting eligibility
        int age = 17;
        String votingStatus = (age >= 18) ? "Can vote" : "Cannot vote yet";
        System.out.println("Age " + age + ": " + votingStatus);

        // Example 2: Absolute value
        int number = -25;
        int absoluteValue = (number < 0) ? -number : number;
        System.out.println("Absolute value of " + number + " is " +
absoluteValue);

        // Example 3: Even/Odd check
        int num = 42;
        String parity = (num % 2 == 0) ? "even" : "odd";
        System.out.println(num + " is " + parity);

        // Example 4: Price discount
        double originalPrice = 150.0;
        boolean isMember = true;
        double finalPrice = isMember ? originalPrice * 0.9 : originalPrice;  //
10% discount for members
        System.out.println("Original price: $" + originalPrice);
        System.out.println("Member discount: " + (isMember ? "Yes" : "No"));
        System.out.println("Final price: $" + finalPrice);

        // Example 5: Plural form
        int itemCount = 1;
        String message = "You have " + itemCount + " item" + (itemCount == 1 ? ""
: "s");
        System.out.println(message);

        itemCount = 5;
        message = "You have " + itemCount + " item" + (itemCount == 1 ? "" : "s");
        System.out.println(message);

        // Example 6: Nested ternary (use sparingly!)
        int temperature = 25;
        String weatherAdvice = (temperature > 30) ? "It's hot - stay hydrated!" :
                               (temperature > 20) ? "Nice weather - enjoy
outdoors!" :
                               (temperature > 10) ? "Cool weather - wear a jacket"
:
                               "Cold weather - dress warmly!";
        System.out.println("Temperature: " + temperature + "°C");
        System.out.println("Advice: " + weatherAdvice);

        // Type compatibility in ternary
        System.out.println("\n=== TYPE COMPATIBILITY ===");

        boolean condition = true;
```

```
        // Both branches must be compatible types
        Object result1 = condition ? "Hello" : "World";        // String and
String
        Object result2 = condition ? "Hello" : 42;             // String and int →
Object
        Number result3 = condition ? 3.14 : 42;                // double and int →
Number

        // This won't compile - incompatible types without common supertype
        // String result4 = condition ? "Hello" : 42;           // ❌ Error

        System.out.println("Result1: " + result1);
        System.out.println("Result2: " + result2);
        System.out.println("Result3: " + result3);

        // When to use ternary operator
        System.out.println("\n=== WHEN TO USE TERNARY ===");
        System.out.println("Use ternary when:");
        System.out.println("✅ Simple condition with two outcomes");
        System.out.println("✅ Assigning values based on condition");
        System.out.println("✅ Short, readable expressions");
        System.out.println("✅ Inline conditional values");
        System.out.println();
        System.out.println("Avoid ternary when:");
        System.out.println("❌ Complex nested conditions");
        System.out.println("❌ Multiple statements needed");
        System.out.println("❌ Side effects in conditions");
        System.out.println("❌ Reduces code readability");

        // Performance note
        System.out.println("\n=== PERFORMANCE ===");
        System.out.println("Ternary operator and if-else have similar
performance.");
        System.out.println("Choose based on readability, not performance.");
    }
}
```

3.3 Control Statements - Loops (20 minutes)

🔄 While Loop

```
public class WhileLoop {
    public static void main(String[] args) {
        System.out.println("=== WHILE LOOP ===");

        // Basic while loop
        System.out.println("Counting from 1 to 5:");
        int count = 1;
        while (count <= 5) {
            System.out.println("Count: " + count);
```

```
            count++;  // Don't forget to increment!
        }

        // While loop with complex condition
        System.out.println("\nPassword attempt simulation:");
        String correctPassword = "secret123";
        String userInput = "wrong";
        int attempts = 0;
        int maxAttempts = 3;

        while (!userInput.equals(correctPassword) && attempts < maxAttempts) {
            attempts++;
            System.out.println("Attempt " + attempts + ": Password incorrect");

            // Simulate different user inputs
            switch (attempts) {
                case 1: userInput = "password"; break;
                case 2: userInput = "123456"; break;
                case 3: userInput = "secret123"; break;
            }
        }

        if (userInput.equals(correctPassword)) {
            System.out.println("Access granted!");
        } else {
            System.out.println("Account locked after " + maxAttempts + " failed
attempts");
        }

        // Infinite loop (be careful!)
        System.out.println("\nInfinite loop prevention:");
        int safetyCounter = 0;
        int x = 1;

        while (x > 0 && safetyCounter < 1000000) {  // Safety check
            x = x * 2;
            safetyCounter++;
            if (safetyCounter % 10 == 0) {
                System.out.println("x = " + x + " (iteration " + safetyCounter +
")");
            }

            // Prevent infinite loop in demo
            if (safetyCounter == 30) {
                System.out.println("Breaking to prevent infinite loop...");
                break;
            }
        }

        // Sum calculation using while loop
        System.out.println("\nSum of numbers 1 to 100:");
        int sum = 0;
        int number = 1;
```

```
        while (number <= 100) {
            sum += number;
            number++;
        }

        System.out.println("Sum = " + sum);

        // Input validation simulation
        System.out.println("\nInput validation (simulated):");
        int userAge = -5;  // Invalid input
        String[] simulatedInputs = {"-5", "abc", "150", "25"};
        int inputIndex = 0;

        while (userAge < 0 || userAge > 120) {
            String input = simulatedInputs[inputIndex++];
            System.out.println("Processing input: " + input);

            try {
                userAge = Integer.parseInt(input);
                if (userAge < 0 || userAge > 120) {
                    System.out.println("Age must be between 0 and 120");
                }
            } catch (NumberFormatException e) {
                System.out.println("Please enter a valid number");
                userAge = -1;  // Keep loop running
            }
        }

        System.out.println("Valid age entered: " + userAge);
    }
}
```

🔁 Do-While Loop

```
public class DoWhileLoop {
    public static void main(String[] args) {
        System.out.println("=== DO-WHILE LOOP ===");

        // Basic do-while loop
        System.out.println("Basic do-while example:");
        int i = 1;
        do {
            System.out.println("Iteration: " + i);
            i++;
        } while (i <= 3);

        // Key difference: executes at least once
        System.out.println("\nDifference between while and do-while:");

        // While loop with false condition
        System.out.println("While loop with false condition:");
```

```
        int x = 10;
        while (x < 5) {
            System.out.println("This will never print");
            x++;
        }
        System.out.println("While loop completed without executing body");

        // Do-while loop with false condition
        System.out.println("\nDo-while loop with false condition:");
        int y = 10;
        do {
            System.out.println("This prints once even though condition is false");
            y++;
        } while (y < 5);

        // Menu system example (classic use case for do-while)
        System.out.println("\n=== MENU SYSTEM SIMULATION ===");
        int choice = 0;
        int menuIteration = 0;
        int[] simulatedChoices = {1, 3, 2, 4};  // Simulated user inputs

        do {
            System.out.println("\n--- Restaurant Menu ---");
            System.out.println("1. Order Pizza");
            System.out.println("2. Order Burger");
            System.out.println("3. Order Salad");
            System.out.println("4. Exit");
            System.out.print("Enter your choice: ");

            // Simulate user input
            if (menuIteration < simulatedChoices.length) {
                choice = simulatedChoices[menuIteration];
                System.out.println(choice);
                menuIteration++;
            } else {
                choice = 4;  // Exit
            }

            switch (choice) {
                case 1:
                    System.out.println("🍕 Pizza ordered! That'll be $12.99");
                    break;
                case 2:
                    System.out.println("🍔 Burger ordered! That'll be $8.99");
                    break;
                case 3:
                    System.out.println("🥗 Salad ordered! That'll be $6.99");
                    break;
                case 4:
                    System.out.println("Thank you for visiting! Goodbye!");
                    break;
                default:
                    System.out.println("Invalid choice. Please try again.");
            }
```

```
        } while (choice != 4);

        // Game loop simulation
        System.out.println("\n=== GAME LOOP SIMULATION ===");
        boolean gameRunning = true;
        int playerHealth = 100;
        int round = 1;

        do {
            System.out.println("\n--- Round " + round + " ---");
            System.out.println("Player Health: " + playerHealth);

            // Simulate game events
            int damage = (int)(Math.random() * 30) + 10;  // Random damage 10-39
            playerHealth -= damage;
            System.out.println("You took " + damage + " damage!");

            if (playerHealth <= 0) {
                System.out.println("💀 Game Over! You survived " + round + "
rounds.");
                gameRunning = false;
            } else if (round == 5) {
                System.out.println("🎉 You won! Survived all rounds!");
                gameRunning = false;
            } else {
                // Player gets some health back
                int healing = (int)(Math.random() * 20) + 5;  // Random healing 5-
24
                playerHealth = Math.min(100, playerHealth + healing);
                System.out.println("You healed " + healing + " health points!");
            }

            round++;
        } while (gameRunning);

        // Number guessing game
        System.out.println("\n=== NUMBER GUESSING GAME ===");
        int secretNumber = 42;  // In real game, this would be random
        int guess = 0;
        int attempts = 0;
        int[] simulatedGuesses = {25, 60, 40, 45, 42};  // Simulated player
guesses
        int guessIndex = 0;

        System.out.println("I'm thinking of a number between 1 and 100...");

        do {
            attempts++;

            // Simulate player guess
            if (guessIndex < simulatedGuesses.length) {
                guess = simulatedGuesses[guessIndex++];
            } else {
                guess = secretNumber;  // Ensure game ends
```

```
            }

            System.out.println("Attempt " + attempts + ": Your guess is " +
guess);

            if (guess < secretNumber) {
                System.out.println("Too low! Try higher.");
            } else if (guess > secretNumber) {
                System.out.println("Too high! Try lower.");
            } else {
                System.out.println("🎯 Congratulations! You guessed it in " +
attempts + " attempts!");
            }

        } while (guess != secretNumber && attempts < 10);

        if (guess != secretNumber) {
            System.out.println("😔 Sorry, you've run out of attempts. The number
was " + secretNumber);
        }

        // When to use do-while vs while
        System.out.println("\n=== WHEN TO USE DO-WHILE ===");
        System.out.println("Use DO-WHILE when:");
        System.out.println("✅ Loop body must execute at least once");
        System.out.println("✅ Menu systems");
        System.out.println("✅ Input validation");
        System.out.println("✅ Game loops");
        System.out.println("✅ 'Do something, then check if continue' pattern");
        System.out.println();
        System.out.println("Use WHILE when:");
        System.out.println("✅ Loop might not need to execute at all");
        System.out.println("✅ Condition should be checked before first
execution");
        System.out.println("✅ More common loop pattern");
    }
}
```

🔢 For Loop

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

        // Basic for loop structure
        System.out.println("Basic for loop (counting 1 to 5):");
        for (int i = 1; i <= 5; i++) {
            System.out.println("Count: " + i);
        }

        // For loop components explained
```

```
        System.out.println("\nFor loop components:");
        System.out.println("for (initialization; condition; increment/decrement)
{");
        System.out.println("    // loop body");
        System.out.println("}");

        // Different increment patterns
        System.out.println("\nDifferent increment patterns:");

        // Increment by 2
        System.out.println("Even numbers 0 to 10:");
        for (int i = 0; i <= 10; i += 2) {
            System.out.print(i + " ");
        }
        System.out.println();

        // Decrement
        System.out.println("Countdown from 5 to 1:");
        for (int i = 5; i >= 1; i--) {
            System.out.print(i + " ");
        }
        System.out.println("🚀 Blast off!");

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

        // Nested for loops
        System.out.println("\n=== NESTED FOR LOOPS ===");

        // Multiplication table
        System.out.println("Multiplication table (1-5):");
        for (int i = 1; i <= 5; i++) {
            for (int j = 1; j <= 5; j++) {
                System.out.printf("%3d", i * j);
            }
            System.out.println();
        }

        // Pattern printing
        System.out.println("\nStar patterns:");

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

        // Inverted triangle
```

```
        System.out.println("Inverted triangle:");
        for (int i = 5; i >= 1; i--) {
            for (int j = 1; j <= i; j++) {
                System.out.print("* ");
            }
            System.out.println();
        }

        // Pyramid
        System.out.println("Pyramid:");
        for (int i = 1; i <= 5; i++) {
            // Print spaces
            for (int j = 1; j <= 5 - i; j++) {
                System.out.print(" ");
            }
            // Print stars
            for (int j = 1; j <= 2 * i - 1; j++) {
                System.out.print("*");
            }
            System.out.println();
        }

        // Practical examples
        System.out.println("\n=== PRACTICAL EXAMPLES ===");

        // Sum of numbers
        System.out.println("Sum of numbers 1 to 100:");
        int sum = 0;
        for (int i = 1; i <= 100; i++) {
            sum += i;
        }
        System.out.println("Sum = " + sum);

        // Factorial calculation
        System.out.println("\nFactorial of 5:");
        int factorial = 1;
        int n = 5;
        for (int i = 1; i <= n; i++) {
            factorial *= i;
            System.out.println(i + "! = " + factorial);
        }

        // Prime number check
        System.out.println("\nChecking if numbers 2-20 are prime:");
        for (int num = 2; num <= 20; num++) {
            boolean isPrime = true;

            for (int i = 2; i <= Math.sqrt(num); i++) {
                if (num % i == 0) {
                    isPrime = false;
                    break;  // No need to check further
                }
            }
```

```
            if (isPrime) {
                System.out.print(num + " ");
            }
        }
        System.out.println("(prime numbers)");

        // Array processing preview
        System.out.println("\nArray processing with for loop:");
        int[] numbers = {10, 25, 33, 47, 52, 68, 71, 89, 94};

        System.out.println("Array elements:");
        for (int i = 0; i < numbers.length; i++) {
            System.out.println("Index " + i + ": " + numbers[i]);
        }

        // Find maximum in array
        int max = numbers[0];
        for (int i = 1; i < numbers.length; i++) {
            if (numbers[i] > max) {
                max = numbers[i];
            }
        }
        System.out.println("Maximum value: " + max);

        // Enhanced for loop (for-each) preview
        System.out.println("\nEnhanced for loop (for-each):");
        System.out.println("Array elements using for-each:");
        for (int number : numbers) {
            System.out.print(number + " ");
        }
        System.out.println();

        // Infinite for loop (be careful!)
        System.out.println("\nInfinite for loop prevention:");
        for (int i = 0; i < 1000000; i++) {
            if (i == 5) {
                System.out.println("Breaking at i = " + i + " to prevent infinite
output");
                break;
            }
            System.out.println("i = " + i);
        }

        // Empty for loop components
        System.out.println("\nFlexible for loop syntax:");

        // Initialization outside
        int counter = 0;
        for (; counter < 3; counter++) {
            System.out.println("Counter: " + counter);
        }

        // Manual increment
        for (int i = 0; i < 3; ) {
```

```
            System.out.println("Manual increment: " + i);
            i++;
        }

        // Complex conditions
        System.out.println("\nComplex conditions:");
        for (int i = 1, j = 10; i < j && i + j < 15; i++, j--) {
            System.out.println("i = " + i + ", j = " + j + ", sum = " + (i + j));
        }
    }
}
```

🎯 SESSION 4: ADVANCED CONTROL & ARRAYS

4.1 Advanced Loop Concepts (20 minutes)

🔄 Break and Continue Statements

```
public class BreakContinueStatements {
    public static void main(String[] args) {
        System.out.println("=== BREAK AND CONTINUE STATEMENTS ===");

        // Break statement in loops
        System.out.println("BREAK statement examples:");

        // Break in for 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 immediately
            }
        }

        // Break in while loop
        System.out.println("\nPassword cracking simulation:");
        String[] passwords = {"password", "123456", "admin", "secret", "letmein"};
        String targetPassword = "secret";
        int attempts = 0;

        while (attempts < passwords.length) {
            String currentTry = passwords[attempts];
            attempts++;
            System.out.println("Attempt " + attempts + ": " + currentTry);

            if (currentTry.equals(targetPassword)) {
                System.out.println("Password cracked!");
                break;  // Stop trying once found
            }
```

```
        }

        // Continue statement in loops
        System.out.println("\nCONTINUE statement examples:");

        // Continue in for loop - 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 rest of loop body for even numbers
            }
            System.out.print(i + " ");
        }
        System.out.println();

        // Continue in while loop - input validation
        System.out.println("\nProcessing valid numbers (skip negatives):");
        int[] numbers = {5, -3, 8, -1, 12, -7, 15, 20};
        int index = 0;
        int sum = 0;
        int count = 0;

        while (index < numbers.length) {
            int current = numbers[index];
            index++;

            if (current < 0) {
                System.out.println("Skipping negative number: " + current);
                continue;  // Skip negative numbers
            }

            sum += current;
            count++;
            System.out.println("Added: " + current);
        }

        System.out.println("Sum of positive numbers: " + sum);
        System.out.println("Count of positive numbers: " + count);

        // Nested loops with break and continue
        System.out.println("\n=== NESTED LOOPS WITH BREAK/CONTINUE ===");

        // Break in nested loops (only breaks inner loop)
        System.out.println("Break in nested loops:");
        for (int i = 1; i <= 3; i++) {
            System.out.println("Outer loop i = " + i);
            for (int j = 1; j <= 5; j++) {
                if (j == 3) {
                    System.out.println("  Breaking inner loop at j = " + j);
                    break;  // Only breaks inner loop
                }
                System.out.println("  Inner loop j = " + j);
            }
        }
```

```
        // Continue in nested loops
        System.out.println("\nContinue in nested loops:");
        for (int i = 1; i <= 3; i++) {
            System.out.println("Outer loop i = " + i);
            for (int j = 1; j <= 5; j++) {
                if (j == 3) {
                    System.out.println("  Skipping j = " + j);
                    continue;  // Skip rest of inner loop body
                }
                System.out.println("  Inner loop j = " + j);
            }
        }
    }
}
```

Labeled Break and Continue

```
public class LabeledBreakContinue {
    public static void main(String[] args) {
        System.out.println("=== LABELED BREAK AND CONTINUE ===");

        // Labeled break - breaking out of nested loops
        System.out.println("Labeled break example:");

        outer: for (int i = 1; i <= 3; i++) {
            System.out.println("Outer loop i = " + i);

            for (int j = 1; j <= 5; j++) {
                if (i == 2 && j == 3) {
                    System.out.println("  Breaking out of both loops at i=" + i +
", j=" + j);
                    break outer;  // Breaks out of the labeled outer loop
                }
                System.out.println("  Inner loop j = " + j);
            }
            System.out.println("End of outer loop iteration " + i);
        }
        System.out.println("After labeled break");

        // Labeled continue - continuing outer loop from inner loop
        System.out.println("\nLabeled continue example:");

        outerLoop: for (int i = 1; i <= 3; i++) {
            System.out.println("Outer loop i = " + i);

            for (int j = 1; j <= 5; j++) {
                if (j == 3) {
                    System.out.println("  Continuing outer loop from j = " + j);
                    continue outerLoop;  // Continues the labeled outer loop
                }
```

```
                System.out.println("  Inner loop j = " + j);
            }
            System.out.println("This won't print when continue outerLoop is
executed");
        }

        // Practical example: Matrix search
        System.out.println("\n=== PRACTICAL EXAMPLE: MATRIX SEARCH ===");

        int[][] matrix = {
            {1, 5, 9, 12},
            {2, 6, 10, 15},
            {3, 7, 11, 18},
            {4, 8, 13, 20}
        };

        int target = 11;
        boolean found = false;
        int foundRow = -1, foundCol = -1;

        searchMatrix: for (int row = 0; row < matrix.length; row++) {
            for (int col = 0; col < matrix[row].length; col++) {
                System.out.println("Checking matrix[" + row + "][" + col + "] = "
+ matrix[row][col]);

                if (matrix[row][col] == target) {
                    found = true;
                    foundRow = row;
                    foundCol = col;
                    System.out.println("Found " + target + " at position (" + row
+ ", " + col + ")");
                    break searchMatrix;  // Break out of both loops
                }
            }
        }

        if (!found) {
            System.out.println("Target " + target + " not found in matrix");
        }

        // Prime factorization example
        System.out.println("\n=== PRIME FACTORIZATION EXAMPLE ===");

        int number = 60;
        System.out.println("Prime factorization of " + number + ":");

        factorization: for (int factor = 2; factor <= number; factor++) {
            while (number % factor == 0) {
                System.out.print(factor + " ");
                number /= factor;

                if (number == 1) {
                    break factorization;  // All factors found
                }
```

```
            }
        }
        System.out.println("\nFactorization complete");

        // Menu system with labeled break
        System.out.println("\n=== MENU SYSTEM WITH LABELED BREAK ===");

        boolean exitProgram = false;
        String[][] menuChoices = {
            {"1", "2", "3"},  // Main menu choices
            {"1", "1", "2"},  // Submenu choices
            {"4", "4", "4"}   // Exit choices
        };

        int choiceIndex = 0;

        programLoop: while (!exitProgram) {
            System.out.println("\n--- Main Menu ---");
            System.out.println("1. Settings");
            System.out.println("2. Games");
            System.out.println("3. Help");
            System.out.println("4. Exit");

            String mainChoice = menuChoices[0][choiceIndex % 3];
            System.out.println("Choice: " + mainChoice);

            switch (mainChoice) {
                case "1":
                    settingsMenu: while (true) {
                        System.out.println("\n  --- Settings Menu ---");
                        System.out.println("  1. Display Settings");
                        System.out.println("  2. Back to Main Menu");

                        String settingsChoice = menuChoices[1][choiceIndex % 3];
                        System.out.println("  Choice: " + settingsChoice);

                        switch (settingsChoice) {
                            case "1":
                                System.out.println("  Display settings
configured");
                                break;
                            case "2":
                                System.out.println("  Returning to main menu");
                                break settingsMenu;  // Break out of settings menu
                            default:
                                System.out.println("  Invalid choice");
                        }
                        choiceIndex++;

                        // Prevent infinite demo loop
                        if (choiceIndex > 2) break settingsMenu;
                    }
                    break;
```

```
                case "2":
                    System.out.println("Game menu would go here");
                    break;

                case "3":
                    System.out.println("Help information displayed");
                    break;

                case "4":
                    System.out.println("Goodbye!");
                    break programLoop;  // Exit the entire program

                default:
                    System.out.println("Invalid choice");
            }

            choiceIndex++;
            // Prevent infinite demo loop
            if (choiceIndex >= 3) exitProgram = true;
        }

        System.out.println("\n=== BEST PRACTICES ===");
        System.out.println("• Use labeled break/continue sparingly");
        System.out.println("• Consider refactoring into methods instead");
        System.out.println("• Label names should be descriptive");
        System.out.println("• Avoid deeply nested labeled structures");
        System.out.println("• Document complex control flow clearly");
    }
}
```

🔄 Enhanced For Loop (For-Each)

```
public class EnhancedForLoop {
    public static void main(String[] args) {
        System.out.println("=== ENHANCED FOR LOOP (FOR-EACH) ===");

        // Basic enhanced for loop with arrays
        System.out.println("Basic for-each with array:");
        int[] numbers = {10, 20, 30, 40, 50};

        // Traditional for loop
        System.out.println("Traditional for loop:");
        for (int i = 0; i < numbers.length; i++) {
            System.out.println("Index " + i + ": " + numbers[i]);
        }

        // Enhanced for loop (for-each)
        System.out.println("\nEnhanced for loop (for-each):");
        for (int number : numbers) {
            System.out.println("Value: " + number);
        }
```

```java
        // For-each with different data types
        System.out.println("\n=== FOR-EACH WITH DIFFERENT TYPES ===");

        // String array
        String[] fruits = {"Apple", "Banana", "Orange", "Grape", "Mango"};
        System.out.println("Fruits in basket:");
        for (String fruit : fruits) {
            System.out.println("- " + fruit);
        }

        // Character array
        char[] vowels = {'a', 'e', 'i', 'o', 'u'};
        System.out.println("\nVowels:");
        for (char vowel : vowels) {
            System.out.print(vowel + " ");
        }
        System.out.println();

        // Double array
        double[] prices = {19.99, 25.50, 12.75, 8.99, 45.00};
        System.out.println("\nProduct prices:");
        double total = 0;
        for (double price : prices) {
            System.out.printf("$%.2f ", price);
            total += price;
        }
        System.out.printf("\nTotal: $%.2f\n", total);

        // Multi-dimensional arrays
        System.out.println("\n=== FOR-EACH WITH 2D ARRAYS ===");

        int[][] matrix = {
            {1, 2, 3},
            {4, 5, 6},
            {7, 8, 9}
        };

        System.out.println("Matrix using for-each:");
        for (int[] row : matrix) {           // Each element is a 1D array
            for (int element : row) {        // Each element in the row
                System.out.print(element + " ");
            }
            System.out.println();
        }

        // String 2D array
        String[][] schedule = {
            {"Math", "Science", "English"},
            {"History", "Art", "PE"},
            {"Music", "Computer", "Study"}
        };

        System.out.println("\nClass schedule:");
```

```
        String[] periods = {"Period 1", "Period 2", "Period 3"};
        int dayNum = 1;

        for (String[] day : schedule) {
            System.out.println("Day " + dayNum + ":");
            int periodNum = 0;
            for (String subject : day) {
                System.out.println("  " + periods[periodNum] + ": " + subject);
                periodNum++;
            }
            dayNum++;
        }

        // Limitations of for-each loop
        System.out.println("\n=== LIMITATIONS OF FOR-EACH ===");

        int[] data = {1, 2, 3, 4, 5};

        // ❌ Cannot modify array elements
        System.out.println("Original array:");
        for (int value : data) {
            System.out.print(value + " ");
        }
        System.out.println();

        // This doesn't modify the original array
        System.out.println("\nTrying to modify (won't work):");
        for (int value : data) {
            value = value * 2;  // This only modifies the local variable
        }

        System.out.println("Array after 'modification':");
        for (int value : data) {
            System.out.print(value + " ");  // Still original values
        }
        System.out.println();

        // ✅ Use traditional for loop to modify
        System.out.println("\nCorrect way to modify array:");
        for (int i = 0; i < data.length; i++) {
            data[i] = data[i] * 2;
        }

        for (int value : data) {
            System.out.print(value + " ");
        }
        System.out.println();

        // ❌ Cannot access index in for-each
        System.out.println("\nNeed index? Use traditional for loop:");
        for (int i = 0; i < fruits.length; i++) {
            System.out.println("Index " + i + ": " + fruits[i]);
        }
```

```
        // ❌ Cannot iterate backwards
        System.out.println("\nIterating backwards (traditional for loop):");
        for (int i = fruits.length - 1; i >= 0; i--) {
            System.out.println(fruits[i]);
        }

        // ❌ Cannot skip elements easily
        System.out.println("\nSkipping every other element (traditional for
loop):");
        for (int i = 0; i < numbers.length; i += 2) {
            System.out.print(numbers[i] + " ");
        }
        System.out.println();

        // When to use for-each vs traditional for loop
        System.out.println("\n=== WHEN TO USE FOR-EACH VS TRADITIONAL FOR ===");

        // Perfect for for-each: simple iteration
        System.out.println("Finding maximum value (perfect for for-each):");
        int[] scores = {85, 92, 78, 96, 88, 91};
        int maxScore = scores[0];

        for (int score : scores) {
            if (score > maxScore) {
                maxScore = score;
            }
        }
        System.out.println("Maximum score: " + maxScore);

        // Better with traditional for: need index
        System.out.println("\nFinding index of maximum value (need traditional
for):");
        maxScore = scores[0];
        int maxIndex = 0;

        for (int i = 0; i < scores.length; i++) {
            if (scores[i] > maxScore) {
                maxScore = scores[i];
                maxIndex = i;
            }
        }
        System.out.println("Maximum score: " + maxScore + " at index " +
maxIndex);

        System.out.println("\n=== BEST PRACTICES ===");
        System.out.println("Use FOR-EACH when:");
        System.out.println("✅ Simply reading/processing all elements");
        System.out.println("✅ Don't need to modify original data");
        System.out.println("✅ Don't need element indices");
        System.out.println("✅ Code readability is priority");
        System.out.println("✅ Iterating through collections");
        System.out.println();
        System.out.println("Use TRADITIONAL FOR when:");
        System.out.println("✅ Need to modify array elements");
```

```
        System.out.println("✅ Need element indices");
        System.out.println("✅ Need to iterate backwards");
        System.out.println("✅'A' + 1 = " + b);  // Prints 'B'

        // Character properties
        System.out.println("\nCharacter properties:");
        System.out.println("Is 'A' a letter? " + Character.isLetter('A'));
        System.out.println("Is '5' a digit? " + Character.isDigit('5'));
        System.out.println("Is ' ' whitespace? " + Character.isWhitespace(' '));
    }
}
```

🏠 Reference Data Types

📝 String Type

```
public class StringDemo {
    public static void main(String[] args) {
        // String creation
        String greeting = "Hello, World!";           // String literal
        String name = new String("Java");            // String object
        String empty = "";                           // Empty string
        String nullString = null;                    // null reference

        System.out.println("Greeting: " + greeting);
        System.out.println("Name: " + name);
        System.out.println("Empty string length: " + empty.length());

        // String operations
        String firstName = "John";
        String lastName = "Doe";
        String fullName = firstName + " " + lastName;  // Concatenation

        System.out.println("\nString operations:");
        System.out.println("Full name: " + fullName);
        System.out.println("Length: " + fullName.length());
        System.out.println("Uppercase: " + fullName.toUpperCase());
        System.out.println("Lowercase: " + fullName.toLowerCase());
        System.out.println("Starts with 'John'? " + fullName.startsWith("John"));
        System.out.println("Contains 'Doe'? " + fullName.contains("Doe"));

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

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

```
        System.out.println("str1.equals(str3): " + str1.equals(str3));   // true
(same content)

        // String immutability
        String original = "Java";
        String modified = original.concat(" Programming");
        System.out.println("\nString immutability:");
        System.out.println("Original: " + original);    // Still "Java"
        System.out.println("Modified: " + modified);    // "Java Programming"
    }
}
```

Arrays (Preview)

```
public class ArrayPreview {
    public static void main(String[] args) {
        // Array declaration and initialization
        int[] numbers = {1, 2, 3, 4, 5};
        String[] colors = {"Red", "Green", "Blue"};

        System.out.println("First number: " + numbers[0]);
        System.out.println("Second color: " + colors[1]);
        System.out.println("Array length: " + numbers.length);

        // Arrays are reference types
        int[] copy = numbers;  // Same reference, not a copy
        copy[0] = 100;
        System.out.println("Original array first element: " + numbers[0]);  //
Also changed!
    }
}
```

Variable Declaration and Initialization

Variable Declaration Syntax

```
public class VariableDeclaration {
    // Instance variables (belong to object)
    private int instanceVar;
    private String instanceString = "Default";

    // Static variables (belong to class)
    static int staticVar = 100;
    static final double PI = 3.14159;  // Constant

    public static void main(String[] args) {
        // Local variables (declared inside methods)
        int localVar;                   // Declaration only
```

```
        // System.out.println(localVar); // ❌ Error: not initialized

        int initializedVar = 42;         // Declaration with initialization
        System.out.println(initializedVar);  // ✅ OK

        // Multiple variable declaration
        int a, b, c;                     // Multiple declaration
        int x = 1, y = 2, z = 3;        // Multiple declaration with
initialization

        // Variable reassignment
        int number = 10;
        System.out.println("Initial: " + number);
        number = 20;                     // Reassignment
        System.out.println("After reassignment: " + number);

        // Final variables (constants)
        final int CONSTANT = 100;
        // CONSTANT = 200;               // ❌ Error: cannot reassign final
variable

        System.out.println("Constant value: " + CONSTANT);
    }
}
```

Variable Naming Rules and Conventions

```
public class VariableNaming {
    public static void main(String[] args) {
        // ✅ Valid variable names
        int age = 25;
        String firstName = "John";       // camelCase (conventional)
        double account_balance = 1000.0; // snake_case (less common in Java)
        char _underscore = 'x';          // Starting with underscore
        int $dollar = 100;               // Starting with dollar sign
        String name2 = "Test";           // Containing numbers

        // ✅ Valid but not conventional
        String Δ = "Delta";              // Unicode characters
        int मेराचर = 5;                  // Non-English characters

        // ❌ Invalid variable names (compilation errors)
        // int 2name = "Test";           // Cannot start with number
        // String class = "MyClass";     // 'class' is a keyword
        // double my-variable = 3.14;    // Hyphen not allowed
        // int my variable = 10;         // Space not allowed

        // 🎯 Java naming conventions
        String userName = "john_doe";        // Variables: camelCase
        final double MAX_VALUE = 100.0;      // Constants: UPPER_SNAKE_CASE
```

```
        System.out.println("All variables declared successfully!");
    }
}
```

🔍 Variable Scope and Lifetime

```
public class VariableScope {
    // Class-level variables
    static int staticVariable = 1;       // Static scope - shared by all instances
    int instanceVariable = 2;           // Instance scope - unique per object

    public static void main(String[] args) {
        // Local variables
        int localVariable = 3;           // Method scope

        System.out.println("Static variable: " + staticVariable);
        // System.out.println("Instance variable: " + instanceVariable); // ❌
Error: need object
        System.out.println("Local variable: " + localVariable);

        // Block scope
        if (true) {
            int blockVariable = 4;       // Block scope
            System.out.println("Block variable: " + blockVariable);
            System.out.println("Local variable in block: " + localVariable);  //
✅ Accessible
        }
        // System.out.println("Block variable: " + blockVariable); // ❌ Error:
out of scope

        // Loop scope
        for (int i = 0; i < 3; i++) {   // 'i' has loop scope
            int loopVariable = i * 2;    // New loopVariable for each iteration
            System.out.println("Loop iteration " + i + ": " + loopVariable);
        }
        // System.out.println("Loop variable: " + i); // ❌ Error: out of scope
    }

    public void instanceMethod() {
        System.out.println("Instance variable in method: " + instanceVariable);
// ✅ Accessible
        System.out.println("Static variable in method: " + staticVariable);     //
✅ Accessible
        // System.out.println("Local variable: " + localVariable);              //
❌ Error: out of scope
    }
}
```

💡 Data Type Conversion Preview

```
public class TypeConversionPreview {
    public static void main(String[] args) {
        // Implicit conversion (widening)
        int intValue = 100;
        long longValue = intValue;       // int → long (automatic)
        double doubleValue = longValue;  // long → double (automatic)

        System.out.println("Implicit conversion:");
        System.out.println("int: " + intValue);
        System.out.println("long: " + longValue);
        System.out.println("double: " + doubleValue);

        // Explicit conversion (narrowing) - covered in detail in Session 2
        double pi = 3.14159;
        int truncated = (int) pi;        // Explicit cast required

        System.out.println("\nExplicit conversion:");
        System.out.println("double: " + pi);
        System.out.println("int (truncated): " + truncated);
    }
}
```

🎯 SESSION 2: OPERATORS & TYPE CASTING

2.1 Java Operators (35 minutes)

🔢 Arithmetic Operators

```
public class ArithmeticOperators {
    public static void main(String[] args) {
        int a = 15, b = 4;

        System.out.println("=== ARITHMETIC OPERATORS ===");
        System.out.println("a = " + a + ", b = " + b);

        // Basic arithmetic
        int addition = a + b;           // Addition
        int subtraction = a - b;        // Subtraction
        int multiplication = a * b;     // Multiplication
        int division = a / b;           // Division (integer division)
        int remainder = a % b;          // Modulus (remainder)

        System.out.println("a + b = " + addition);        // 19
        System.out.println("a - b = " + subtraction);     // 11
        System.out.println("a * b = " + multiplication);  // 60
        System.out.println("a / b = " + division);        // 3 (not 3.75!)
        System.out.println("a % b = " + remainder);       // 3
```

```
        // Floating-point division
        double preciseDiv = (double) a / b;
        System.out.println("Precise division: " + preciseDiv);  // 3.75

        // Modulus with negative numbers
        System.out.println("\nModulus with negatives:");
        System.out.println("15 % 4 = " + (15 % 4));     // 3
        System.out.println("-15 % 4 = " + (-15 % 4));   // -3
        System.out.println("15 % -4 = " + (15 % -4));   // 3
        System.out.println("-15 % -4 = " + (-15 % -4)); // -3

        // Division by zero
        try {
            // int error = a / 0;  // Runtime error: ArithmeticException
        } catch (ArithmeticException e) {
            System.out.println("Division by zero error: " + e.getMessage());
        }

        // Floating-point division by zero
        double floatDiv = 5.0 / 0.0;
        System.out.println("5.0 / 0.0 = " + floatDiv);  // Infinity
    }
}
```

Assignment Operators

```
public class AssignmentOperators {
    public static void main(String[] args) {
        System.out.println("=== ASSIGNMENT OPERATORS ===");

        // Simple assignment
        int x = 10;
        System.out.println("x = " + x);

        // Compound assignment operators
        x += 5;  // x = x + 5
        System.out.println("x += 5: " + x);  // 15

        x -= 3;  // x = x - 3
        System.out.println("x -= 3: " + x);  // 12

        x *= 2;  // x = x * 2
        System.out.println("x *= 2: " + x);  // 24

        x /= 4;  // x = x / 4
        System.out.println("x /= 4: " + x);  // 6

        x %= 4;  // x = x % 4
        System.out.println("x %= 4: " + x);  // 2

        // Bitwise compound assignment
```

```
        x = 12;  // Binary: 1100
        x &= 10; // Binary: 1010, Result: 1000 = 8
        System.out.println("x &= 10: " + x);  // 8

        x |= 5;  // Binary: 0101, Result: 1101 = 13
        System.out.println("x |= 5: " + x);   // 13

        x ^= 3;  // Binary: 0011, Result: 1110 = 14
        System.out.println("x ^= 3: " + x);   // 14

        x <<= 1; // Left shift by 1, Result: 11100 = 28
        System.out.println("x <<= 1: " + x);  // 28

        x >>= 2; // Right shift by 2, Result: 111 = 7
        System.out.println("x >>= 2: " + x);  // 7
    }
}
```

🔄 Increment and Decrement Operators

```
public class IncrementDecrementOperators {
    public static void main(String[] args) {
        System.out.println("=== INCREMENT/DECREMENT OPERATORS ===");

        int a = 5;

        // Pre-increment (++variable)
        System.out.println("Original a: " + a);           // 5
        System.out.println("Pre-increment ++a: " + ++a);  // 6 (increment first,
then use)
        System.out.println("After pre-increment: " + a);  // 6

        // Post-increment (variable++)
        a = 5;  // Reset
        System.out.println("\nOriginal a: " + a);         // 5
        System.out.println("Post-increment a++: " + a++); // 5 (use first, then
increment)
        System.out.println("After post-increment: " + a); // 6

        // Pre-decrement (--variable)
        a = 5;  // Reset
        System.out.println("\nOriginal a: " + a);         // 5
        System.out.println("Pre-decrement --a: " + --a);  // 4 (decrement first,
then use)
        System.out.println("After pre-decrement: " + a);  // 4

        // Post-decrement (variable--)
        a = 5;  // Reset
        System.out.println("\nOriginal a: " + a);         // 5
        System.out.println("Post-decrement a--: " + a--); // 5 (use first, then
decrement)
```

```
        System.out.println("After post-decrement: " + a); // 4

        // Complex expressions
        int x = 10, y = 20;
        int result1 = ++x + y++;  // (11) + (20), then y becomes 21
        System.out.println("\n++x + y++ where x=10, y=20");
        System.out.println("Result: " + result1);  // 31
        System.out.println("x: " + x + ", y: " + y);  // x=11, y=21

        // Common pitfall
        x = 5;
        int result2 = x++ + ++x;  // 5 + 7 = 12 (x becomes 6, then 7)
        System.out.println("\nx++ + ++x where x=5");
        System.out.println("Result: " + result2);  // 12
        System.out.println("Final x: " + x);       // 7
    }
}
```

🔍 Relational Operators

```
public class RelationalOperators {
    public static void main(String[] args) {
        System.out.println("=== RELATIONAL OPERATORS ===");

        int a = 10, b = 20, c = 10;

        // Comparison operators
        System.out.println("a = " + a + ", b = " + b + ", c = " + c);

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

        // Floating-point comparisons (be careful!)
        double d1 = 0.1 + 0.2;
        double d2 = 0.3;
        System.out.println("\nFloating-point comparison:");
        System.out.println("0.1 + 0.2 = " + d1);
        System.out.println("0.3 = " + d2);
        System.out.println("(0.1 + 0.2) == 0.3: " + (d1 == d2));  // false!
(precision issue)

        // Better floating-point comparison
        double epsilon = 1e-10;
        boolean isEqual = Math.abs(d1 - d2) < epsilon;
        System.out.println("Safe comparison: " + isEqual);  // true
```

```
        // String comparison
        String str1 = "Hello";
        String str2 = "Hello";
        String str3 = new String("Hello");

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

Logical Operators

```
public class LogicalOperators {
    public static void main(String[] args) {
        System.out.println("=== LOGICAL OPERATORS ===");

        boolean p = true, q = false;

        System.out.println("p = " + p + ", q = " + q);

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

        // Logical OR (||) - Short-circuit evaluation
        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

        // Short-circuit evaluation demonstration
        System.out.println("\nShort-circuit evaluation:");
        int x = 5;

        // In &&, if first condition is false, second is not evaluated
        if (false && (++x > 0)) {
            System.out.println("This won't print");
        }
        System.out.println("x after false &&: " + x);  // 5 (not incremented)

        // In ||, if first condition is true, second is not evaluated
        if (true || (++x > 0)) {
            System.out.println("This will print");
        }
```

```
        System.out.println("x after true ||: " + x);  // 5 (not incremented)

        // Complex logical expressions
        int age = 25;
        boolean hasLicense = true;
        boolean hasInsurance = false;

        boolean canDrive = (age >= 18) && hasLicense && hasInsurance;
        System.out.println("\nCan drive? " + canDrive);  // false

        boolean canRent = (age >= 21) && hasLicense;
        System.out.println("Can rent car? " + canRent);  // true

        // Bitwise logical operators (no short-circuit)
        System.out.println("\nBitwise logical operators:");
        boolean result1 = true & false;   // false (both sides evaluated)
        boolean result2 = true | false;   // true (both sides evaluated)
        boolean result3 = true ^ false;   // true (XOR)

        System.out.println("true & false: " + result1);
        System.out.println("true | false: " + result2);
        System.out.println("true ^ false: " + result3);
    }
}
```

Bitwise Operators

```
public class BitwiseOperators {
    public static void main(String[] args) {
        System.out.println("=== BITWISE OPERATORS ===");

        int a = 12;  // Binary: 1100
        int b = 10;  // Binary: 1010

        System.out.println("a = " + a + " (binary: " + Integer.toBinaryString(a) +
")");
        System.out.println("b = " + b + " (binary: " + Integer.toBinaryString(b) +
")");

        // Bitwise AND (&)
        int and = a & b;  // 1100 & 1010 = 1000 = 8
        System.out.println("a & b = " + and + " (binary: " +
Integer.toBinaryString(and) + ")");

        // Bitwise OR (|)
        int or = a | b;   // 1100 | 1010 = 1110 = 14
        System.out.println("a | b = " + or + " (binary: " +
Integer.toBinaryString(or) + ")");

        // Bitwise XOR (^)
        int xor = a ^ b;  // 1100 ^ 1010 = 0110 = 6
```

```
        System.out.println("a ^ b = " + xor + " (binary: " +
Integer.toBinaryString(xor) + ")");

        // Bitwise NOT (~)
        int not = ~a;     // ~1100 = ...11110011 (two's complement)
        System.out.println("~a = " + not + " (binary: " +
Integer.toBinaryString(not) + ")");

        // Left shift (<<)
        int leftShift = a << 2;  // 1100 << 2 = 110000 = 48
        System.out.println("a << 2 = " + leftShift + " (binary: " +
Integer.toBinaryString(leftShift) + ")");

        // Right shift (>>)
        int rightShift = a >> 2; // 1100 >> 2 = 11 = 3
        System.out.println("a >> 2 = " + rightShift + " (binary: " +
Integer.toBinaryString(rightShift) + ")");

        // Unsigned right shift (>>>)
        int negativeNum = -8;
        int unsignedRightShift = negativeNum >>> 2;
        System.out.println("\nNegative number right shift:");
        System.out.println(negativeNum + " >> 2 = " + (negativeNum >> 2));
        System.out.println(negativeNum + " >>> 2 = " + unsignedRightShift);

        // Practical applications
        System.out.println("\nPractical applications:");

        // Check if number is even/odd using bitwise AND
        int num = 23;
        boolean isEven = (num & 1) == 0;
        System.out.println(num + " is even: " + isEven);

        // Multiply/divide by powers of 2 using shifts
        int multiply = num << 3;  // Multiply by 8 (2^3)
        int divide = num >> 2;    // Divide by 4 (2^2)
        System.out.println(num + " * 8 = " + multiply);
        System.out.println(num + " / 4 = " + divide);

        // Toggle a bit using XOR
        int original = 5;         // Binary: 101
        int mask = 2;             // Binary: 010 (toggle 2nd bit)
        int toggled = original ^ mask;  // Result: 111 = 7
        System.out.println("Toggle bit: " + original + " ^ " + mask + " = " +
toggled);
    }
}
```

Ternary (Conditional) Operator

```java
public class TernaryOperator {
    public static void main(String[] args) {
        System.out.println("=== TERNARY OPERATOR ===");

        // Basic syntax: condition ? valueIfTrue : valueIfFalse
        int a = 10, b = 20;

        int max = (a > b) ? a : b;
        System.out.println("Maximum of " + a + " and " + b + " is: " + max);

        int min = (a < b) ? a : b;
        System.out.println("Minimum of " + a + " and " + b + " is: " + min);

        // String result
        String result = (a > b) ? "a is greater" : "b is greater or equal";
        System.out.println(result);

        // Nested ternary operators (use sparingly!)
        int x = 15;
        String category = (x < 10) ? "small" :
                          (x < 20) ? "medium" : "large";
        System.out.println(x + " is " + category);

        // Equivalent if-else
        String categoryIfElse;
        if (x < 10) {
            categoryIfElse = "small";
        } else if (x < 20) {
            categoryIfElse = "medium";
        } else {
            categoryIfElse = "large";
        }
        System.out.println("If-else equivalent: " + categoryIfElse);

        // Practical examples
        int age = 17;
        String permission = (age >= 18) ? "Can vote" : "Cannot vote";
        System.out.println("Age " + age + ": " + permission);

        double grade = 85.5;
        char letterGrade = (grade >= 90) ? 'A' :
                           (grade >= 80) ? 'B' :
                           (grade >= 70) ? 'C' :
                           (grade >= 60) ? 'D' : 'F';
        System.out.println("Grade " + grade + " = " + letterGrade);

        // Type must be compatible
        int score = 95;
        // This won't compile - different types
        // Object mixed = (score > 90) ? "Excellent" : 95;

        // This works - same type or compatible types
        Object mixed = (score > 90) ? "Excellent" : "Good";
```

```
        System.out.println("Mixed result: " + mixed);
    }
}
```

🔍 Operator Precedence and Associativity

```
public class OperatorPrecedence {
    public static void main(String[] args) {
        System.out.println("=== OPERATOR PRECEDENCE ===");

        // Arithmetic operator precedence
        int result1 = 2 + 3 * 4;           // 2 + 12 = 14 (not 20)
        int result2 = (2 + 3) * 4;         // 5 * 4 = 20
        System.out.println("2 + 3 * 4 = " + result1);
        System.out.println("(2 + 3) * 4 = " + result2);

        // Mixed operators
        int a = 10, b = 5, c = 2;
        int result3 = a + b * c;           // 10 + 10 = 20
        int result4 = a - b / c;           // 10 - 2 = 8
        System.out.println("10 + 5 * 2 = " + result3);
        System.out.println("10 - 5 / 2 = " + result4);

        // Relational and logical precedence
        boolean result5 = 5 > 3 && 2 < 4;  // true && true = true
        boolean result6 = 5 > 3 || 2 > 4;  // true || false = true
        System.out.println("5 > 3 && 2 < 4 = " + result5);
        System.out.println("5 > 3 || 2 > 4 = " + result6);

        // Assignment operator precedence (right-to-left associativity)
        int x, y, z;
        x = y = z = 10;  // Equivalent to: x = (y = (z = 10))
        System.out.println("x = y = z = 10: x=" + x + ", y=" + y + ", z=" + z);

        // Increment/decrement precedence
        int i = 5;
        int result7 = ++i * 2;  // 6 * 2 = 12
        System.out.println("++i * 2 where i=5: " + result7 + ", i=" + i);

        i = 5;
        int result8 = i++ * 2;  // 5 * 2 = 10, then i becomes 6
        System.out.println("i++ * 2 where i=5: " + result8 + ", i=" + i);

        // Complex expression
        int complex = 2 + 3 * 4 > 10 && 5 < 6 ? 100 : 200;
        // Evaluation: 2 + 12 > 10 && 5 < 6 ? 100 : 200
        //             14 > 10 && true ? 100 : 200
        //             true && true ? 100 : 200
        //             true ? 100 : 200
        //             100
        System.out.println("Complex expression result: " + complex);
```

```
        // Use parentheses for clarity
        int clear = ((2 + (3 * 4)) > 10) && (5 < 6) ? 100 : 200;
        System.out.println("Same expression with parentheses: " + clear);
    }
}
```

2.2 Type Casting & Conversion (25 minutes)

🔄 Implicit Type Conversion (Widening)

```
public class ImplicitConversion {
    public static void main(String[] args) {
        System.out.println("=== IMPLICIT TYPE CONVERSION (WIDENING) ===");

        // Numeric widening conversions (automatic)
        byte byteVal = 100;
        short shortVal = byteVal;      // byte → short
        int intVal = shortVal;         // short → int
        long longVal = intVal;         // int → long
        float floatVal = longVal;      // long → float
        double doubleVal = floatVal;   // float → double

        System.out.println("Widening conversion chain:");
        System.out.println("byte: " + byteVal);
        System.out.println("short: " + shortVal);
        System.out.println("int: " + intVal);
        System.out.println("long: " + longVal);
        System.out.println("float: " + floatVal);
        System.out.println("double: " + doubleVal);

        // Character to numeric conversion
        char charVal = 'A';            // ASCII value 65
        int charToInt = charVal;       // char → int
        long charToLong = charVal;     // char → long
        float charToFloat = charVal;   // char → float
        double charToDouble = charVal; // char → double

        System.out.println("\nCharacter conversion:");
        System.out.println("char 'A': " + charVal);
        System.out.println("as int: " + charToInt);
        System.out.println("as long: " + charToLong);
        System.out.println("as float: " + charToFloat);
        System.out.println("as double: " + charToDouble);

        // Mixed arithmetic promotions
        byte b1 = 10, b2 = 20;
        // byte result = b1 + b2;      // ❌ Error: result is promoted to int
        int result = b1 + b2;          // ✅ OK: explicit int declaration
```

```
        short s1 = 100;
        int i1 = 200;
        long l1 = s1 + i1;              // Result promoted to long

        float f1 = 3.14f;
        double d1 = f1 + i1;            // Result promoted to double

        System.out.println("\nArithmetic promotions:");
        System.out.println("byte + byte = int: " + result);
        System.out.println("short + int = long: " + l1);
        System.out.println("float + int = double: " + d1);

        // String concatenation with automatic conversion
        String str = "Number: " + intVal;  // int → String (automatic)
        System.out.println("\nString concatenation: " + str);
    }
}
```

🎯 Explicit Type Conversion (Narrowing)

```
public class ExplicitConversion {
    public static void main(String[] args) {
        System.out.println("=== EXPLICIT TYPE CONVERSION (NARROWING) ===");

        // Numeric narrowing conversions (manual casting required)
        double doubleVal = 123.456;
        float floatVal = (float) doubleVal;    // double → float
        long longVal = (long) floatVal;        // float → long
        int intVal = (int) longVal;            // long → int
        short shortVal = (short) intVal;       // int → short
        byte byteVal = (byte) shortVal;        // short → byte

        System.out.println("Narrowing conversion chain:");
        System.out.println("double: " + doubleVal);
        System.out.println("float: " + floatVal);
        System.out.println("long: " + longVal);
        System.out.println("int: " + intVal);
        System.out.println("short: " + shortVal);
        System.out.println("byte: " + byteVal);

        // Precision loss in floating-point conversion
        double precise = 3.99999999;
        int truncated = (int) precise;         // Truncates decimal part
        System.out.println("\nPrecision loss:");
        System.out.println("double: " + precise);
        System.out.println("int (truncated): " + truncated);

        // Overflow in narrowing conversion
        int bigInt = 300;
        byte smallByte = (byte) bigInt;        // Overflow!
        System.out.println("\nOverflow example:");
```

```
        System.out.println("int: " + bigInt);
        System.out.println("byte (overflow): " + smallByte);  // -44 (due to
overflow)

        // Understanding the overflow
        System.out.println("Explanation: 300 % 256 = " + (300 % 256) +
                           ", but byte is signed, so 44 - 128 = " + (44 - 128));

        // Numeric to character conversion
        int asciiValue = 65;
        char character = (char) asciiValue;    // int → char
        System.out.println("\nNumeric to character:");
        System.out.println("ASCII " + asciiValue + " = '" + character + "'");

        // Character to numeric conversion
        char letter = 'Z';
        int letterValue = (int) letter;        // char → int (implicit, but shown
for clarity)
        System.out.println("Character '" + letter + "' = " + letterValue);

        // Boolean cannot be cast to other types
        boolean flag = true;
        // int boolInt = (int) flag;           // ❌ Error: boolean cannot be
cast

        // String to numeric conversion (using wrapper classes)
        String numberStr = "123";
        int parsedInt = Integer.parseInt(numberStr);        // String → int
        double parsedDouble = Double.parseDouble("45.67");  // String → double

        System.out.println("\nString to numeric conversion:");
        System.out.println("String \"123\" → int: " + parsedInt);
        System.out.println("String \"45.67\" → double: " + parsedDouble);

        // Numeric to String conversion
        int number = 456;
        String numberString = String.valueOf(number);       // int → String
        String anotherWay = "" + number;                    // int → String
(concatenation)

        System.out.println("int 456 → String: \"" + numberString + "\"");
        System.out.println("Alternative: \"" + anotherWay + "\"");
    }
}
```

⚠ Type Casting Pitfalls and Best Practices

```
public class CastingPitfalls {
    public static void main(String[] args) {
        System.out.println("=== TYPE CASTING PITFALLS ===");
```

```
        // 1. Precision loss in floating-point to integer conversion
        double pi = 3.14159;
        int intPi = (int) pi;
        System.out.println("π = " + pi + " → int = " + intPi + " (precision
lost)");

        // Better approach: rounding
        int roundedPi = (int) Math.round(pi);
        System.out.println("Rounded π = " + roundedPi);

        // 2. Overflow in narrowing conversions
        int maxByte = 127;
        int beyondByte = 128;

        byte safeCast = (byte) maxByte;     // Safe
        byte overflowCast = (byte) beyondByte;  // Overflow!

        System.out.println("\nOverflow demonstration:");
        System.out.println("127 → byte = " + safeCast);
        System.out.println("128 → byte = " + overflowCast + " (overflow!)");

        // Safe casting with range checking
        int valueToCheck = 200;
        if (valueToCheck >= Byte.MIN_VALUE && valueToCheck <= Byte.MAX_VALUE) {
            byte safeByte = (byte) valueToCheck;
            System.out.println("Safe cast: " + safeByte);
        } else {
            System.out.println("Value " + valueToCheck + " is outside byte
range");
        }

        // 3. Unexpected results with mixed arithmetic
        int intResult = 5 / 2;              // Integer division
        double doubleResult = 5 / 2;        // Still integer division!
        double correctResult = 5.0 / 2;     // Floating-point division
        double anotherWay = (double) 5 / 2; // Explicit cast

        System.out.println("\nDivision pitfalls:");
        System.out.println("5 / 2 as int: " + intResult);
        System.out.println("5 / 2 as double: " + doubleResult + " (wrong!)");
        System.out.println("5.0 / 2: " + correctResult);
        System.out.println("(double) 5 / 2: " + anotherWay);

        // 4. String parsing exceptions
        try {
            String invalidNumber = "abc123";
            int parsed = Integer.parseInt(invalidNumber);  //
NumberFormatException
        } catch (NumberFormatException e) {
            System.out.println("\nParsing error: " + e.getMessage());
        }

        // Safe string parsing
        String maybeNumber = "456";
```

```
        try {
            int safelyParsed = Integer.parseInt(maybeNumber);
            System.out.println("Safely parsed: " + safelyParsed);
        } catch (NumberFormatException e) {
            System.out.println("Could not parse: " + maybeNumber);
        }

        // 5. Char arithmetic surprises
        char c1 = 'A';
        char c2 = 'B';
        // char sum = c1 + c2;               // ❌ Error: result is int
        int charSum = c1 + c2;              // ✅ OK
        char charSumCast = (char)(c1 + c2); // ✅ Explicit cast

        System.out.println("\nCharacter arithmetic:");
        System.out.println("
```