Practicals

Practical01

```java
// Practical01.java - Basic Java program demonstrating output methods

public class Practical01 {
    public static void main(String[] args) {
        System.out.println("Demonstrating Different Output Methods in Java:\n");

        // 1. Using println() - prints and moves to next line
        System.out.println("1. Using println():");
        System.out.println("Hello, World!");
        System.out.println("This is a new line");
        System.out.println();

        // 2. Using print() - prints without moving to next line
        System.out.println("2. Using print():");
        System.out.print("Hello ");
        System.out.print("World ");
        System.out.print("without line breaks");
        System.out.println("\n");

        // 3. Using printf() - formatted output
        System.out.println("3. Using printf():");
        String name = "Student";
        int age = 20;
        double height = 5.9;
        System.out.printf("Name: %s, Age: %d, Height: %.1f feet%n", name, age, height);
        System.out.println();

        // 4. Demonstrating escape sequences
        System.out.println("4. Using Escape Sequences:");
        System.out.println("Using tab:\tAfter tab");
        System.out.println("Using new line:\nAfter new line");
        System.out.println("Using single quote: \'Hello\'");
        System.out.println("Using double quote: \"World\"");
        System.out.println("Using backslash: \\");
    }
}
```

Practical02

```java
// Practical02.java - Find maximum of three numbers using conditional operator

import java.util.Scanner;

public class Practical02 {
    // Method to find maximum using conditional operator
    public static int findMax(int a, int b, int c) {
        return (a > b) ? ((a > c) ? a : c) : ((b > c) ? b : c);
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.println("Program to Find Maximum of Three Numbers:\n");

        // 1. Using hardcoded values
        System.out.println("1. Testing with hardcoded values:");
        int x = 25, y = 45, z = 15;
        System.out.printf("Numbers are: %d, %d, %d%n", x, y, z);
        System.out.println("Maximum number is: " + findMax(x, y, z));
        System.out.println();

        // 2. Taking user input
        System.out.println("2. Testing with user input:");
        System.out.print("Enter first number: ");
        int num1 = scanner.nextInt();

        System.out.print("Enter second number: ");
        int num2 = scanner.nextInt();

        System.out.print("Enter third number: ");
        int num3 = scanner.nextInt();

        int max = findMax(num1, num2, num3);
        System.out.printf("Maximum number among %d, %d and %d is: %d%n",
                          num1, num2, num3, max);

        // 3. Additional test cases
        System.out.println("\n3. Testing with special cases:");

        // When all numbers are same
        System.out.println("When all numbers are same:");
        System.out.println("Max of (5, 5, 5): " + findMax(5, 5, 5));

        // When two numbers are same
        System.out.println("When two numbers are same:");
        System.out.println("Max of (7, 7, 3): " + findMax(7, 7, 3));

        // With negative numbers
        System.out.println("With negative numbers:");
        System.out.println("Max of (-5, -2, -8): " + findMax(-5, -2, -8));

        scanner.close();
    }
}
```

Practical03

```java
// Practical03.java - Reverse digits of a number using while loop

import java.util.Scanner;

public class Practical03 {
    // Method to reverse digits of a number
    public static int reverseNumber(int num) {
        int reversed = 0;
        boolean isNegative = num < 0;
        num = Math.abs(num);

        while (num > 0) {
            int digit = num % 10;
            reversed = reversed * 10 + digit;
            num /= 10;
        }

        return isNegative ? -reversed : reversed;
    }

    // Method to display the reversal process
    public static void showReversalProcess(int num) {
        System.out.println("\nReversal Process:");
        int temp = Math.abs(num);
        System.out.print("Digits extracted: ");

        // Store digits in array for proper display order
        int[] digits = new int[10];  // Assuming number won't exceed 10 digits
        int count = 0;

        while (temp > 0) {
            digits[count++] = temp % 10;
            temp /= 10;
        }

        // Display digits in order of extraction
        for (int i = 0; i < count; i++) {
            System.out.print(digits[i]);
            if (i < count - 1) {
                System.out.print(", ");
            }
        }
        System.out.println();
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.println("Program to Reverse Digits of a Number:\n");

        // 1. Using hardcoded values
        System.out.println("1. Testing with hardcoded values:");
        int[] testNumbers = {12345, -9876, 1000, 7};

        for (int num : testNumbers) {
            System.out.println("\nOriginal number: " + num);
            showReversalProcess(num);
            System.out.println("Reversed number: " + reverseNumber(num));
        }

        // 2. Taking user input
        System.out.println("\n2. Testing with user input:");
        System.out.print("Enter a number to reverse: ");
        int userNum = scanner.nextInt();

        System.out.println("Original number: " + userNum);
        showReversalProcess(userNum);
        System.out.println("Reversed number: " + reverseNumber(userNum));

        // 3. Special cases demonstration
        System.out.println("\n3. Special cases:");

        // Number ending with zeros
        int numWithZeros = 12000;
        System.out.println("\nNumber ending with zeros: " + numWithZeros);
        showReversalProcess(numWithZeros);
        System.out.println("Reversed number: " + reverseNumber(numWithZeros));

        // Single digit number
        int singleDigit = 5;
        System.out.println("\nSingle digit number: " + singleDigit);
        showReversalProcess(singleDigit);
        System.out.println("Reversed number: " + reverseNumber(singleDigit));

        scanner.close();
    }
}
```

Practical04

```java
// Practical04.java - Add two 3x3 matrices

import java.util.Scanner;

public class Practical04 {
    // Method to input matrix elements
    public static void inputMatrix(int[][] matrix, Scanner scanner, String matrixName) {
        System.out.println("Enter elements for " + matrixName + " (3x3):");
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                System.out.printf("Enter element [%d][%d]: ", i, j);
                matrix[i][j] = scanner.nextInt();
            }
        }
        System.out.println();
    }

    // Method to display matrix
    public static void displayMatrix(int[][] matrix, String matrixName) {
        System.out.println(matrixName + ":");
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                System.out.printf("%4d ", matrix[i][j]);
            }
            System.out.println();
        }
        System.out.println();
    }

    // Method to add two matrices
    public static int[][] addMatrices(int[][] matrix1, int[][] matrix2) {
        int[][] result = new int[3][3];
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                result[i][j] = matrix1[i][j] + matrix2[i][j];
            }
        }
        return result;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.println("Program to Add Two 3x3 Matrices:\n");

        // 1. Using hardcoded matrices
        System.out.println("1. Testing with hardcoded matrices:");
        int[][] matrix1 = {
            {1, 2, 3},
            {4, 5, 6},
            {7, 8, 9}
        };

        int[][] matrix2 = {
            {9, 8, 7},
            {6, 5, 4},
            {3, 2, 1}
        };

        displayMatrix(matrix1, "First Matrix");
        displayMatrix(matrix2, "Second Matrix");

        int[][] result1 = addMatrices(matrix1, matrix2);
        displayMatrix(result1, "Result Matrix (Hardcoded)");
        // 2. Taking user input
        System.out.println("2. Testing with user input:");
        int[][] userMatrix1 = new int[3][3];
        int[][] userMatrix2 = new int[3][3];

        inputMatrix(userMatrix1, scanner, "First Matrix");
        inputMatrix(userMatrix2, scanner, "Second Matrix");

        System.out.println("Entered matrices:");
        displayMatrix(userMatrix1, "First Matrix");
        displayMatrix(userMatrix2, "Second Matrix");

        int[][] result2 = addMatrices(userMatrix1, userMatrix2);
        displayMatrix(result2, "Result Matrix (User Input)");

        // 3. Special case demonstration
        System.out.println("3. Special case - Adding zero matrix:");
        int[][] zeroMatrix = new int[3][3];  // All elements are 0 by default

        displayMatrix(matrix1, "Original Matrix");
        displayMatrix(zeroMatrix, "Zero Matrix");

        int[][] result3 = addMatrices(matrix1, zeroMatrix);
        displayMatrix(result3, "Result Matrix (Adding Zero Matrix)");

        scanner.close();
    }
}
```

Practical05

```
// Practical05.java - Generate first n prime numbers

import java.util.Scanner;

public class Practical05 {
    // Method to check if a number is prime
    public static boolean isPrime(int number) {
        if (number < 2) {
            return false;
        }
        for (int i = 2; i <= Math.sqrt(number); i++) {
            if (number % i == 0) {
                return false;
            }
        }
        return true;
    }

    // Method to generate first n prime numbers
    public static void generatePrimes(int n) {
        if (n <= 0) {
            System.out.println("Please enter a positive number.");
            return;
        }

        System.out.println("First " + n + " prime numbers are:");
        int count = 0;
        int number = 2;

        while (count < n) {
            if (isPrime(number)) {
                System.out.print(number);
                count++;

                // Add formatting
                if (count < n) {
                    System.out.print(", ");
                }
                if (count % 10 == 0) {
                    System.out.println();
                }
            }
            number++;
        }
        System.out.println();
    }

    // Method to show prime checking process
    public static void demonstratePrimeCheck(int number) {
        System.out.printf("\nChecking if %d is prime:%n", number);
        if (number < 2) {
            System.out.println(number + " is not prime (less than 2)");
            return;
        }

        for (int i = 2; i <= Math.sqrt(number); i++) {
            System.out.printf("Checking divisibility by %d: ", i);
            if (number % i == 0) {
                System.out.printf("%d is divisible by %d, so it's not prime%n", number,
i);
                return;
            }
            System.out.println("Not divisible");
        }
        System.out.println(number + " is prime");
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.println("Program to Generate First N Prime Numbers:\n");
```

```java
        // 1. Using hardcoded value
        System.out.println("1. Testing with hardcoded value (n=10):");
        generatePrimes(10);

        // 2. Demonstrate prime checking process
        System.out.println("\n2. Demonstrating prime checking process:");
        int[] testNumbers = {7, 12, 23, 35};
        for (int num : testNumbers) {
            demonstratePrimeCheck(num);
        }

        // 3. Taking user input
        System.out.println("\n3. Testing with user input:");
        System.out.print("Enter how many prime numbers you want to generate: ");
        int n = scanner.nextInt();
        generatePrimes(n);

        // 4. Handle special cases
        System.out.println("\n4. Testing special cases:");
        System.out.println("Generating 0 prime numbers:");
        generatePrimes(0);

        System.out.println("\nGenerating 1 prime number:");
        generatePrimes(1);

        scanner.close();
    }
}
```

Practical06

```java
// Practical06.java - Create Student class and demonstrate object creation

class Student {
    private String enrollmentNo;
    private String name;

    // Constructor
    public Student(String enrollmentNo, String name) {
        this.enrollmentNo = enrollmentNo;
        this.name = name;
    }

    // Getter methods
    public String getEnrollmentNo() {
        return enrollmentNo;
    }

    public String getName() {
        return name;
    }

    // Setter methods
    public void setEnrollmentNo(String enrollmentNo) {
        this.enrollmentNo = enrollmentNo;
    }

    public void setName(String name) {
        this.name = name;
    }

    // Method to display student details
    public void displayDetails() {
        System.out.println("Student Details:");
        System.out.println("Enrollment No: " + enrollmentNo);
        System.out.println("Name: " + name);
        System.out.println();
    }
}

public class Practical06 {
    public static void main(String[] args) {
        System.out.println("Demonstrating Student Class and Objects:\n");

        // 1. Creating three student objects
        System.out.println("1. Creating and displaying three students:");
        Student student1 = new Student("A101", "John Smith");
        Student student2 = new Student("A102", "Emma Watson");
        Student student3 = new Student("A103", "Michael Johnson");

        // Display student information
        student1.displayDetails();
        student2.displayDetails();
        student3.displayDetails();

        // 2. Demonstrating getter methods
        System.out.println("2. Using getter methods:");
        System.out.println("Student 1:");
        System.out.println("Enrollment No: " + student1.getEnrollmentNo());
        System.out.println("Name: " + student1.getName());
        System.out.println();

        // 3. Demonstrating setter methods
        System.out.println("3. Using setter methods to update student details:");
        System.out.println("Updating student2's information:");
        student2.setName("Emma Thompson");
        System.out.println("After update:");
        student2.displayDetails();

        // 4. Creating array of students
        System.out.println("4. Working with array of students:");
        Student[] students = {
            new Student("B101", "Alice Brown"),
            new Student("B102", "Bob Wilson"),
            new Student("B103", "Carol White")
        };

        System.out.println("Displaying all students in array:");
        for (Student student : students) {
            student.displayDetails();
        }
    }
}
```

Practical07

```
// Practical07.java - Rectangle class with constructor initialization

class Rectangle {
    private double height;
    private double width;

    // Default constructor
    public Rectangle() {
        this.height = 0.0;
        this.width = 0.0;
    }

    // Parameterized constructor
    public Rectangle(double height, double width) {
        this.height = height;
        this.width = width;
    }

    // Copy constructor
    public Rectangle(Rectangle other) {
        this.height = other.height;
        this.width = other.width;
    }

    // Getter methods
    public double getHeight() {
        return height;
    }

    public double getWidth() {
        return width;
    }

    // Setter methods
    public void setHeight(double height) {
        this.height = height;
    }
```

```java
    public void setWidth(double width) {
        this.width = width;
    }

    // Method to calculate area
    public double calculateArea() {
        return height * width;
    }

    // Method to calculate perimeter
    public double calculatePerimeter() {
        return 2 * (height + width);
    }

    // Method to display rectangle details
    public void displayDetails() {
        System.out.println("Rectangle Details:");
        System.out.printf("Height: %.2f units%n", height);
        System.out.printf("Width: %.2f units%n", width);
        System.out.printf("Area: %.2f square units%n", calculateArea());
        System.out.printf("Perimeter: %.2f units%n", calculatePerimeter());
        System.out.println();
    }
}

public class Practical07 {
    public static void main(String[] args) {
        System.out.println("Demonstrating Rectangle Class with Constructors:\n");

        // 1. Using default constructor
        System.out.println("1. Creating rectangle using default constructor:");
        Rectangle rect1 = new Rectangle();
        rect1.displayDetails();

        // 2. Using parameterized constructor
        System.out.println("2. Creating rectangle using parameterized constructor:");
        Rectangle rect2 = new Rectangle(5.0, 3.0);
        rect2.displayDetails();

        // 3. Using copy constructor
        System.out.println("3. Creating rectangle using copy constructor:");
        Rectangle rect3 = new Rectangle(rect2);
        System.out.println("Copied rectangle details:");
        rect3.displayDetails();

        // 4. Demonstrating setter methods
        System.out.println("4. Using setter methods:");
        rect1.setHeight(4.0);
        rect1.setWidth(6.0);
        System.out.println("After setting new dimensions:");
        rect1.displayDetails();
        // 5. Demonstrating getter methods
        System.out.println("5. Using getter methods:");
        System.out.printf("Rectangle 2 height: %.2f units%n", rect2.getHeight());
        System.out.printf("Rectangle 2 width: %.2f units%n", rect2.getWidth());
        System.out.println();

        // 6. Array of rectangles
        System.out.println("6. Working with array of rectangles:");
        Rectangle[] rectangles = {
            new Rectangle(2.0, 3.0),
            new Rectangle(4.0, 4.0),
            new Rectangle(3.0, 5.0)
        };

        System.out.println("Details of all rectangles:");
        for (Rectangle rect : rectangles) {
            rect.displayDetails();
        }

        // 7. Find rectangle with largest area
        System.out.println("7. Finding rectangle with largest area:");
        Rectangle maxAreaRect = rectangles[0];
        for (Rectangle rect : rectangles) {
            if (rect.calculateArea() > maxAreaRect.calculateArea()) {
                maxAreaRect = rect;
            }
        }
        System.out.println("Rectangle with largest area:");
        maxAreaRect.displayDetails();
    }
}
```

Practical08

```
// Practical08.java - Demonstrate use of 'this' keyword

public class Practical08 {
    private int number;
    private String text;

    // Constructor using 'this' to distinguish parameters from instance variables
    public Practical08(int number, String text) {
        this.number = number;
        this.text = text;
    }

    // Method using 'this' to call another method of current object
    public void display() {
        System.out.println("Number: " + this.number);
        System.out.println("Text: " + this.text);
        this.showMore();  // Using 'this' to call another method
    }

    // Method using 'this' to pass current object as parameter
    public void showMore() {
        System.out.println("Demonstrating method call using 'this'");
        this.processObject(this);  // Passing current object as parameter
    }

    // Method accepting object of same class as parameter
    public void processObject(Practical08 obj) {
        System.out.println("Processing object with number: " + obj.number);
    }

    // Method returning current object using 'this'
    public Practical08 updateNumber(int number) {
        this.number = number;
        return this;  // Method chaining by returning current object
    }

    public static void main(String[] args) {
        System.out.println("Demonstrating the use of 'this' keyword:\n");

        // Creating object and demonstrating various uses of 'this'
        Practical08 obj = new Practical08(42, "Hello");

        System.out.println("Initial object state:");
        obj.display();

        System.out.println("\nDemonstrating method chaining using 'this':");
        obj.updateNumber(100).display();

        // Creating another object to show constructor usage of 'this'
        System.out.println("\nCreating another object:");
        Practical08 obj2 = new Practical08(99, "World");
        obj2.display();
    }
}
```

Practical09

```
// Practical09.java - Demonstrate use of 'static' keyword

public class Practical09 {
    // Static variable
    private static int instanceCount = 0;

    // Non-static variables
    private int id;
    private String name;

    // Static constant
    private static final String COLLEGE_NAME = "My College";

    // Static block - executed when class is loaded
    static {
        System.out.println("Static block executed - Class loading...");
        System.out.println("College Name: " + COLLEGE_NAME);
    }

    // Constructor
    public Practical09(String name) {
        this.id = ++instanceCount;
        this.name = name;
    }

    // Static method
    public static int getInstanceCount() {
        return instanceCount;
    }

    // Static method to display college info
    public static void displayCollegeInfo() {
        System.out.println("College Name: " + COLLEGE_NAME);
        System.out.println("Total Students: " + getInstanceCount());
    }

    // Non-static method
    public void displayStudentInfo() {
        System.out.println("Student ID: " + this.id);
        System.out.println("Student Name: " + this.name);
        System.out.println("College: " + COLLEGE_NAME);  // Static variable accessed in
non-static method
    }

    public static void main(String[] args) {
        System.out.println("Demonstrating static keyword usage:\n");

        // Accessing static method before creating any object
        System.out.println("Initial instance count: " + Practical09.getInstanceCount());

        // Creating objects and demonstrating static variable
        System.out.println("\nCreating student objects:");
        Practical09 student1 = new Practical09("John");
        Practical09 student2 = new Practical09("Emma");
        Practical09 student3 = new Practical09("Michael");

        // Displaying individual student information
        System.out.println("\nStudent Information:");
        student1.displayStudentInfo();
        System.out.println();
        student2.displayStudentInfo();
        System.out.println();
```

```java
        student3.displayStudentInfo();

        // Displaying college information using static method
        System.out.println("\nCollege Information:");
        Practical09.displayCollegeInfo();

        // Demonstrating that static variable is shared
        System.out.println("\nFinal instance count: " + Practical09.getInstanceCount());
    }
}
```

Practical10

```java
// Practical10.java - Demonstrate use of 'final' keyword

// Final class - cannot be inherited
final class FinalClass {
    public void display() {
        System.out.println("This class cannot be inherited");
    }
}

class Parent {
    // Final method - cannot be overridden
    final void showMessage() {
        System.out.println("This method cannot be overridden");
    }
}

class Child extends Parent {
    // This would cause error if uncommented:
    // void showMessage() { } // Cannot override final method

    void displayChild() {
        System.out.println("Child class calling parent's final method:");
        showMessage();
    }
}

public class Practical10 {
    // Final variable - must be initialized and cannot be changed
    private final int MAX_VALUE = 100;

    // Final reference variable
    private final StringBuilder builder = new StringBuilder();

    // Final static constant
    private static final double PI = 3.14159;

    // Blank final variable - must be initialized in constructor
    private final String message;
    // Constructor to initialize final variable
    public Practical10(String msg) {
        this.message = msg;  // Initializing blank final variable
    }

    public void demonstrateFinal() {
        // This would cause error:
        // MAX_VALUE = 200; // Cannot modify final variable

        // Can modify object state even though reference is final
        builder.append("Hello ");
        builder.append("World");

        System.out.println("Final variable MAX_VALUE: " + MAX_VALUE);
        System.out.println("Final StringBuilder content: " + builder.toString());
        System.out.println("Final static PI: " + PI);
        System.out.println("Final message: " + message);
    }

    public static void main(String[] args) {
        System.out.println("Demonstrating final keyword usage:\n");

        // Demonstrating final variables
        Practical10 obj = new Practical10("This is a final message");
        obj.demonstrateFinal();

        // Demonstrating final class
        System.out.println("\nDemonstrating final class:");
        FinalClass finalObj = new FinalClass();
        finalObj.display();

        // Demonstrating final method
        System.out.println("\nDemonstrating final method:");
        Child child = new Child();
        child.displayChild();

        // Demonstrating final parameter in lambda expression
        System.out.println("\nDemonstrating final parameter in lambda:");
        Runnable run = () -> {
            final String param = "Hello";
            System.out.println("Parameter cannot be modified: " + param);
        };
        run.run();

        // Demonstrating final local variable
        final int number = 100;
        System.out.println("\nFinal local variable: " + number);
        // This would cause error:
        // number = 200; // Cannot modify final variable
    }
}
```

Practical11

```
// Practical11.java - Demonstrate method overloading with Shape class

public class Practical11 {
    // Class to demonstrate method overloading
    public class Shape {
        // Method to calculate area of circle
        public float area(float radius) {
            return (float) (Math.PI * radius * radius);
        }

        // Overloaded method to calculate area of rectangle
        public float area(float length, float width) {
            return length * width;
        }

        // Additional overloaded methods to show more variations
        public float area(int radius) {
            // Overloaded method with different parameter type
            return (float) (Math.PI * radius * radius);
        }

        public double area(double radius) {
            // Overloaded method with different return type
            return Math.PI * radius * radius;
        }

        public float area(float base, float height, String shape) {
            // Overloaded method for triangle if shape is "triangle"
            if (shape.equalsIgnoreCase("triangle")) {
                return 0.5f * base * height;
            }
            return 0; // Return 0 for invalid shape
        }
    }

    public static void main(String[] args) {
        Practical11 practical = new Practical11();
        Shape shape = practical.new Shape();

        // Test values
        float radius = 5.0f;
        float length = 6.0f;
        float width = 4.0f;
        float base = 8.0f;
        float height = 3.0f;

        System.out.println("Demonstrating Method Overloading:\n");
        // Calculate and display area of circle using float parameter
        System.out.println("Area of Circle (float radius = " + radius + "):");
        System.out.printf("%.2f square units\n\n", shape.area(radius));

        // Calculate and display area of rectangle
        System.out.println("Area of Rectangle (length = " + length + ", width = " + width
+ "):");
        System.out.printf("%.2f square units\n\n", shape.area(length, width));

        // Calculate and display area of circle using int parameter
        System.out.println("Area of Circle (int radius = 5):");
        System.out.printf("%.2f square units\n\n", shape.area(5));

        // Calculate and display area of circle using double parameter
        System.out.println("Area of Circle (double radius = 5.0):");
        System.out.printf("%.2f square units\n\n", shape.area(5.0));

        // Calculate and display area of triangle
        System.out.println("Area of Triangle (base = " + base + ", height = " + height +
"):");
        System.out.printf("%.2f square units\n\n", shape.area(base, height, "triangle"));

        // Demonstrate method selection based on parameter type
        System.out.println("Demonstrating automatic method selection based on parameter
type:");
        System.out.println("Calling area(5.0f) - selects float version: " +
shape.area(5.0f));
        System.out.println("Calling area(5) - selects int version: " + shape.area(5));
        System.out.println("Calling area(5.0) - selects double version: " +
shape.area(5.0));
    }
}
```

Practical12

```
// Practical12.java - Demonstrate constructor overloading

public class Practical12 {
    // Instance variables
    private String name;
    private int age;
    private String city;
    private String occupation;

    // Default constructor
    public Practical12() {
        System.out.println("Default Constructor Called");
        this.name = "Unknown";
        this.age = 0;
        this.city = "Not Specified";
        this.occupation = "Not Specified";
    }

    // Constructor with name parameter
    public Practical12(String name) {
        System.out.println("Constructor with name parameter Called");
        this.name = name;
        this.age = 0;
        this.city = "Not Specified";
        this.occupation = "Not Specified";
    }

    // Constructor with name and age parameters
    public Practical12(String name, int age) {
        System.out.println("Constructor with name and age parameters Called");
        this.name = name;
        this.age = age;
        this.city = "Not Specified";
        this.occupation = "Not Specified";
    }

    // Constructor with all parameters
    public Practical12(String name, int age, String city, String occupation) {
        System.out.println("Constructor with all parameters Called");
        this.name = name;
        this.age = age;
        this.city = city;
        this.occupation = occupation;
    }

    // Constructor using another constructor (Copy constructor)
    public Practical12(Practical12 other) {
        System.out.println("Copy Constructor Called");
        this.name = other.name;
        this.age = other.age;
        this.city = other.city;
        this.occupation = other.occupation;
    }

    // Method to display person details
    public void displayDetails() {
        System.out.println("\nPerson Details:");
        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
        System.out.println("City: " + city);
        System.out.println("Occupation: " + occupation);
    }

    public static void main(String[] args) {
        System.out.println("Demonstrating Constructor Overloading:\n");

        // Creating objects using different constructors
        System.out.println("1. Creating object with default constructor:");
        Practical12 person1 = new Practical12();
        person1.displayDetails();

        System.out.println("\n2. Creating object with name parameter:");
        Practical12 person2 = new Practical12("John");
        person2.displayDetails();

        System.out.println("\n3. Creating object with name and age parameters:");
        Practical12 person3 = new Practical12("Emma", 25);
        person3.displayDetails();

        System.out.println("\n4. Creating object with all parameters:");
        Practical12 person4 = new Practical12("Michael", 30, "New York", "Engineer");
        person4.displayDetails();

        System.out.println("\n5. Creating object using copy constructor:");
        Practical12 person5 = new Practical12(person4);
        person5.displayDetails();
    }
}
```

Practical13

```
// Practical13.java - Demonstrate String class methods

public class Practical13 {
    // Method to demonstrate charAt()
    public static void demonstrateCharAt(String str) {
        System.out.println("\nDemonstrating charAt() method:");
        System.out.println("String: " + str);
        System.out.println("Character at index 0: " + str.charAt(0));
        System.out.println("Character at index 4: " + str.charAt(4));
        System.out.println("Last character: " + str.charAt(str.length() - 1));
    }

    // Method to demonstrate contains()
    public static void demonstrateContains(String str) {
        System.out.println("\nDemonstrating contains() method:");
        System.out.println("String: " + str);
        System.out.println("Contains 'Java'? " + str.contains("Java"));
        System.out.println("Contains 'Python'? " + str.contains("Python"));
        System.out.println("Contains 'programming'? " + str.contains("programming"));
    }

    // Method to demonstrate format()
    public static void demonstrateFormat() {
        System.out.println("\nDemonstrating format() method:");
        String formatted = String.format("Name: %s, Age: %d, Height: %.2f", "John", 25,
5.9);
        System.out.println(formatted);
        // More format examples
        System.out.println(String.format("Binary: %b, Character: %c", true, 'A'));
        System.out.println(String.format("Hex: %x, Scientific: %e", 255, 123.456));
        System.out.println(String.format("Left justified: '%-10s'", "Hello"));
        System.out.println(String.format("Right justified: '%10s'", "Hello"));
    }

    // Method to demonstrate length()
    public static void demonstrateLength(String str) {
        System.out.println("\nDemonstrating length() method:");
        System.out.println("String: " + str);
        System.out.println("Length: " + str.length());

        // Additional length examples
        String empty = "";
        String withSpaces = "   Hello   ";
        System.out.println("Empty string length: " + empty.length());
        System.out.println("String with spaces length: " + withSpaces.length());
    }

    // Method to demonstrate split()
    public static void demonstrateSplit() {
        System.out.println("\nDemonstrating split() method:");

        // Split by space
        String sentence = "Java Programming is fun";
        System.out.println("Original string: " + sentence);
        System.out.println("Splitting by space:");
        String[] words = sentence.split(" ");
        for (int i = 0; i < words.length; i++) {
            System.out.println("Word " + (i + 1) + ": " + words[i]);
        }

        // Split by comma
        String csvData = "John,25,New York,Engineer";
        System.out.println("\nSplitting CSV data:");
        String[] data = csvData.split(",");
        System.out.println("Name: " + data[0]);
        System.out.println("Age: " + data[1]);
        System.out.println("City: " + data[2]);
        System.out.println("Occupation: " + data[3]);
    }

    public static void main(String[] args) {
        System.out.println("Demonstrating String Class Methods:");

        String testString = "Java Programming";

        // Demonstrate all methods
        demonstrateCharAt(testString);
        demonstrateContains(testString);
        demonstrateFormat();
        demonstrateLength(testString);
        demonstrateSplit();
    }
}
```

Practical14

```
// Practical14.java - Demonstrate single inheritance

// Parent class
class Animal {
    protected String name;
    protected int age;

    // Constructor
    public Animal(String name, int age) {
        this.name = name;
        this.age = age;
    }

    // Methods
    public void eat() {
        System.out.println(name + " is eating.");
    }

    public void sleep() {
        System.out.println(name + " is sleeping.");
    }

    public void displayInfo() {
        System.out.println("Name: " + name);
        System.out.println("Age: " + age + " years");
    }
}

// Child class inheriting from Animal
class Dog extends Animal {
    private String breed;

    // Constructor
    public Dog(String name, int age, String breed) {
        super(name, age);  // Call parent constructor
        this.breed = breed;
    }

    // Additional methods specific to Dog
    public void bark() {
        System.out.println(name + " is barking!");
    }

    public void fetch() {
```

```java
        System.out.println(name + " is fetching the ball.");
    }

    // Override parent method
    @Override
    public void displayInfo() {
        super.displayInfo();  // Call parent method
        System.out.println("Breed: " + breed);
    }
}

public class Practical14 {
    public static void main(String[] args) {
        System.out.println("Demonstrating Single Inheritance:\n");

        // Create instances of parent and child classes
        System.out.println("1. Creating Animal object (Parent class):");
        Animal animal = new Animal("Generic Animal", 5);
        animal.displayInfo();
        animal.eat();
        animal.sleep();

        System.out.println("\n2. Creating Dog object (Child class):");
        Dog dog = new Dog("Buddy", 3, "Golden Retriever");

        // Accessing inherited methods
        System.out.println("\nAccessing inherited methods:");
        dog.displayInfo();
        dog.eat();
        dog.sleep();

        // Accessing Dog-specific methods
        System.out.println("\nAccessing Dog-specific methods:");
        dog.bark();
        dog.fetch();

        // Demonstrating polymorphism
        System.out.println("\n3. Demonstrating polymorphism:");
        Animal animalDog = new Dog("Max", 2, "German Shepherd");
        System.out.println("Calling methods on Dog object through Animal reference:");
        animalDog.displayInfo();
        animalDog.eat();
        animalDog.sleep();
        // Note: Can't call bark() or fetch() through Animal reference
    }
}
```

Practical15

```java
// Practical15.java - Demonstrate multilevel inheritance
// Grandparent class
class Vehicle {
    protected String brand;
    protected String model;

    public Vehicle(String brand, String model) {
        this.brand = brand;
        this.model = model;
    }

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

    public void stop() {
        System.out.println("Vehicle is stopping...");
    }

    public void displayInfo() {
        System.out.println("Brand: " + brand);
        System.out.println("Model: " + model);
    }
}

// Parent class inheriting from Vehicle
class Car extends Vehicle {
    private int numDoors;
    private String fuelType;

    public Car(String brand, String model, int numDoors, String fuelType) {
        super(brand, model);
        this.numDoors = numDoors;
        this.fuelType = fuelType;
    }

    public void accelerate() {
        System.out.println("Car is accelerating...");
    }

    public void brake() {
        System.out.println("Car is braking...");
    }

    @Override
    public void displayInfo() {
        super.displayInfo();
        System.out.println("Number of Doors: " + numDoors);
        System.out.println("Fuel Type: " + fuelType);
    }
}

// Child class inheriting from Car
```

```java
class ElectricCar extends Car {
    private int batteryCapacity;
    private int range;

    public ElectricCar(String brand, String model, int numDoors,
                      int batteryCapacity, int range) {
        super(brand, model, numDoors, "Electric");
        this.batteryCapacity = batteryCapacity;
        this.range = range;
    }

    public void charge() {
        System.out.println("Electric car is charging...");
    }

    public void displayBatteryStatus() {
        System.out.println("Battery Status: 75%");
    }

    @Override
    public void displayInfo() {
        super.displayInfo();
        System.out.println("Battery Capacity: " + batteryCapacity + " kWh");
        System.out.println("Range: " + range + " km");
    }
}

public class Practical15 {
    public static void main(String[] args) {
        System.out.println("Demonstrating Multilevel Inheritance:\n");

        // Create instances of all three classes
        System.out.println("1. Creating Vehicle object (Grandparent class):");
        Vehicle vehicle = new Vehicle("Generic", "Basic");
        vehicle.displayInfo();
        vehicle.start();
        vehicle.stop();

        System.out.println("\n2. Creating Car object (Parent class):");
        Car car = new Car("Toyota", "Camry", 4, "Petrol");
        car.displayInfo();
        car.start();  // Inherited from Vehicle
        car.accelerate();  // Car's own method
        car.brake();  // Car's own method
        car.stop();  // Inherited from Vehicle

        System.out.println("\n3. Creating ElectricCar object (Child class):");
        ElectricCar electricCar = new ElectricCar("Tesla", "Model 3", 4, 75, 350);
        electricCar.displayInfo();
        electricCar.start();  // Inherited from Vehicle
        electricCar.accelerate();  // Inherited from Car
        electricCar.charge();  // ElectricCar's own method
        electricCar.displayBatteryStatus();  // ElectricCar's own method
        electricCar.stop();  // Inherited from Vehicle

        // Demonstrating polymorphism
        System.out.println("\n4. Demonstrating polymorphism:");
        Vehicle polymorphicCar = new ElectricCar("Tesla", "Model S", 4, 100, 400);
        System.out.println("Calling methods on ElectricCar through Vehicle reference:");
        polymorphicCar.displayInfo();  // Will call ElectricCar's version
        polymorphicCar.start();
        polymorphicCar.stop();
    }
}
```

Practical16

```
// Practical16.java - Demonstrate hierarchical inheritance

// Parent class
class Employee {
    protected int id;
    protected String name;
    protected double baseSalary;

    public Employee(int id, String name, double baseSalary) {
        this.id = id;
        this.name = name;
        this.baseSalary = baseSalary;
    }

    public void work() {
        System.out.println(name + " is working");
    }

    public double calculateSalary() {
        return baseSalary;
    }

    public void displayInfo() {
        System.out.println("ID: " + id);
        System.out.println("Name: " + name);
        System.out.println("Salary: $" + calculateSalary());
    }
}

// First child class
class Developer extends Employee {
    private String programmingLanguage;
    private double bonus;

    public Developer(int id, String name, double baseSalary,
                    String programmingLanguage, double bonus) {
```

```
        super(id, name, baseSalary);
        this.programmingLanguage = programmingLanguage;
        this.bonus = bonus;
    }

    public void code() {
        System.out.println(name + " is coding in " + programmingLanguage);
    }

    @Override
    public double calculateSalary() {
        return baseSalary + bonus;
    }

    @Override
    public void displayInfo() {
        super.displayInfo();
        System.out.println("Role: Developer");
        System.out.println("Programming Language: " + programmingLanguage);
        System.out.println("Bonus: $" + bonus);
    }
}

// Second child class
class Designer extends Employee {
    private String designTool;
    private int projectsCompleted;

    public Designer(int id, String name, double baseSalary,
                   String designTool, int projectsCompleted) {
        super(id, name, baseSalary);
        this.designTool = designTool;
        this.projectsCompleted = projectsCompleted;
    }

    public void design() {
        System.out.println(name + " is designing using " + designTool);
    }

    @Override
    public double calculateSalary() {
        return baseSalary + (projectsCompleted * 100); // $100 bonus per project
    }

    @Override
    public void displayInfo() {
        super.displayInfo();
        System.out.println("Role: Designer");
        System.out.println("Design Tool: " + designTool);
        System.out.println("Projects Completed: " + projectsCompleted);
    }
}
```

```java
// Third child class
class Manager extends Employee {
    private int teamSize;
    private double managementBonus;

    public Manager(int id, String name, double baseSalary,
                  int teamSize, double managementBonus) {
        super(id, name, baseSalary);
        this.teamSize = teamSize;
        this.managementBonus = managementBonus;
    }

    public void manage() {
        System.out.println(name + " is managing a team of " + teamSize + " people");
    }

    @Override
    public double calculateSalary() {
        return baseSalary + managementBonus + (teamSize * 100); // $100 per team member
    }

    @Override
    public void displayInfo() {
        super.displayInfo();
        System.out.println("Role: Manager");
        System.out.println("Team Size: " + teamSize);
        System.out.println("Management Bonus: $" + managementBonus);
    }
}

public class Practical16 {
    public static void main(String[] args) {
        System.out.println("Demonstrating Hierarchical Inheritance:\n");

        // Creating objects of different employee types
        Developer dev = new Developer(101, "John", 70000, "Java", 5000);
        Designer designer = new Designer(102, "Emma", 65000, "Adobe XD", 5);
        Manager manager = new Manager(103, "Michael", 80000, 8, 10000);

        // Demonstrating Developer
        System.out.println("1. Developer Details:");
        dev.displayInfo();
        dev.work();  // Inherited method
        dev.code();  // Specific method

        // Demonstrating Designer
        System.out.println("\n2. Designer Details:");
        designer.displayInfo();
        designer.work();  // Inherited method
        designer.design();  // Specific method
        // Demonstrating Manager
        System.out.println("\n3. Manager Details:");
        manager.displayInfo();
        manager.work();  // Inherited method
        manager.manage();  // Specific method

        // Demonstrating polymorphism
        System.out.println("\n4. Demonstrating polymorphism:");
        Employee[] employees = {dev, designer, manager};
        for (Employee emp : employees) {
            System.out.println("\nEmployee Information:");
            emp.displayInfo();
        }
    }
}
```

Practical17

```java
// Practical17.java - Demonstrate method overriding

// Parent class
class Shape {
    protected String color;

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

    // Method to be overridden
    public void draw() {
        System.out.println("Drawing a shape");
    }

    public void getInfo() {
        System.out.println("This is a " + color + " shape");
    }

    public double calculateArea() {
        return 0.0;  // Default implementation
    }
}

// First child class
class Circle extends Shape {
    private double radius;

    public Circle(String color, double radius) {
        super(color);
        this.radius = radius;
    }
    // Override draw method
    @Override
    public void draw() {
        System.out.println("Drawing a circle with radius " + radius);
    }

    @Override
    public void getInfo() {
        System.out.println("This is a " + color + " circle with radius " + radius);
    }

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

// Second child class
class Rectangle extends Shape {
    private double length;
    private double width;

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

    @Override
    public void draw() {
        System.out.println("Drawing a rectangle with length " + length + " and width " +
width);
    }

    @Override
    public void getInfo() {
        System.out.println("This is a " + color + " rectangle with length " + length +
                         " and width " + width);
    }

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

// Third child class
class Triangle extends Shape {
    private double base;
    private double height;

    public Triangle(String color, double base, double height) {
```

```java
        super(color);
        this.base = base;
        this.height = height;
    }

    @Override
    public void draw() {
        System.out.println("Drawing a triangle with base " + base + " and height " +
height);
    }

    @Override
    public void getInfo() {
        System.out.println("This is a " + color + " triangle with base " + base +
                          " and height " + height);
    }

    @Override
    public double calculateArea() {
        return 0.5 * base * height;
    }
}

public class Practical17 {
    public static void main(String[] args) {
        System.out.println("Demonstrating Method Overriding:\n");

        // Create objects of different shapes
        Circle circle = new Circle("Red", 5.0);
        Rectangle rectangle = new Rectangle("Blue", 4.0, 6.0);
        Triangle triangle = new Triangle("Green", 3.0, 8.0);

        // Demonstrate method overriding for Circle
        System.out.println("1. Circle:");
        circle.draw();
        circle.getInfo();
        System.out.printf("Area: %.2f square units\n", circle.calculateArea());

        // Demonstrate method overriding for Rectangle
        System.out.println("\n2. Rectangle:");
        rectangle.draw();
        rectangle.getInfo();
        System.out.printf("Area: %.2f square units\n", rectangle.calculateArea());

        // Demonstrate method overriding for Triangle
        System.out.println("\n3. Triangle:");
        triangle.draw();
        triangle.getInfo();
        System.out.printf("Area: %.2f square units\n", triangle.calculateArea());

        // Demonstrate polymorphism with method overriding
        System.out.println("\n4. Demonstrating polymorphism:");
        Shape[] shapes = {circle, rectangle, triangle};
        for (Shape shape : shapes) {
            System.out.println("\nShape details:");
            shape.draw();       // Calls overridden method
            shape.getInfo();    // Calls overridden method
            System.out.printf("Area: %.2f square units\n", shape.calculateArea());
        }
    }
}
```

Practical18

```
// Practical18.java - Demonstrate toString() method overriding

class Car {
    private String name;
    private int topSpeed;
    private String color;
    private double price;

    // Constructor
    public Car(String name, int topSpeed, String color, double price) {
        this.name = name;
        this.topSpeed = topSpeed;
        this.color = color;
        this.price = price;
    }

    // Overriding toString() method
    @Override
    public String toString() {
        return String.format("Car[name=%s, topSpeed=%d mph, color=%s, price=$%.2f]",
                           name, topSpeed, color, price);
    }

    // Getters
    public String getName() {
        return name;
    }

    public int getTopSpeed() {
        return topSpeed;
    }

    public String getColor() {
        return color;
    }

    public double getPrice() {
        return price;
    }
```

```java
}

public class Practical18 {
    // Method to display car details in a formatted way
    public static void displayCarDetails(Car car, int carNumber) {
        System.out.println("Car " + carNumber + " Details:");
        System.out.println("Name: " + car.getName());
        System.out.println("Top Speed: " + car.getTopSpeed() + " mph");
        System.out.println("Color: " + car.getColor());
        System.out.printf("Price: $%.2f\n", car.getPrice());
        System.out.println("toString() output: " + car.toString());
        System.out.println();
    }

    public static void main(String[] args) {
        System.out.println("Demonstrating toString() Method Overriding:\n");

        // Create 5 car instances
        Car car1 = new Car("Tesla Model S", 200, "Red", 89990.00);
        Car car2 = new Car("BMW M3", 180, "Blue", 69900.00);
        Car car3 = new Car("Toyota Supra", 155, "Yellow", 43540.00);
        Car car4 = new Car("Porsche 911", 182, "Black", 101200.00);
        Car car5 = new Car("Ford Mustang", 160, "White", 27205.00);

        // Store cars in an array
        Car[] cars = {car1, car2, car3, car4, car5};

        // Display details of each car
        for (int i = 0; i < cars.length; i++) {
            displayCarDetails(cars[i], i + 1);
        }

        // Demonstrate direct use of toString()
        System.out.println("Direct println() calls (implicitly uses toString()):");
        for (Car car : cars) {
            System.out.println(car);  // println automatically calls toString()
        }

        // Demonstrate toString() in different contexts
        System.out.println("\nDemonstrating toString() in different contexts:");

        // In string concatenation
        String description = "My car is: " + car1;
        System.out.println(description);

        // In StringBuilder
        StringBuilder sb = new StringBuilder();
        sb.append("Available car: ").append(car2);
        System.out.println(sb.toString());

        // In formatted string
        System.out.printf("Featured car: %s%n", car3);
    }
}
```

Practical19

```
// Practical19.java - Demonstrate multiple inheritance using interfaces

// First interface
interface Printer {
    void print();
    void checkInk();
}

// Second interface
interface Scanner {
    void scan();
    void checkScanQuality();
}

// Third interface
interface Fax {
    void fax(String destination);
    void checkFaxLine();
}

// Class implementing multiple interfaces
class AllInOnePrinter implements Printer, Scanner, Fax {
    private String modelName;
    private boolean inkAvailable;
    private boolean scannerWorking;
    private boolean faxLineConnected;

    public AllInOnePrinter(String modelName) {
        this.modelName = modelName;
        this.inkAvailable = true;
        this.scannerWorking = true;
        this.faxLineConnected = true;
    }

    // Implementing Printer interface methods
    @Override
    public void print() {
        System.out.println(modelName + " is printing a document");
    }

    @Override
    public void checkInk() {
        System.out.println("Ink status: " + (inkAvailable ? "Available" : "Low"));
    }

    // Implementing Scanner interface methods
    @Override
    public void scan() {
        System.out.println(modelName + " is scanning a document");
    }

    @Override
    public void checkScanQuality() {
        System.out.println("Scanner status: " +
            (scannerWorking ? "Working properly" : "Needs maintenance"));
    }

    // Implementing Fax interface methods
    @Override
    public void fax(String destination) {
        System.out.println(modelName + " is faxing to " + destination);
    }

    @Override
    public void checkFaxLine() {
        System.out.println("Fax line status: " +
            (faxLineConnected ? "Connected" : "Disconnected"));
    }

    // Additional method specific to AllInOnePrinter
    public void displayStatus() {
        System.out.println("\nDevice Status for " + modelName + ":");
        checkInk();
        checkScanQuality();
        checkFaxLine();
    }
}

// Class implementing only Printer and Scanner interfaces
class BasicPrinter implements Printer, Scanner {
    private String modelName;

    public BasicPrinter(String modelName) {
        this.modelName = modelName;
    }

    @Override
    public void print() {
        System.out.println(modelName + " is printing a document");
    }

    @Override
    public void checkInk() {
        System.out.println("Checking ink levels for " + modelName);
    }

    @Override
    public void scan() {
        System.out.println(modelName + " is scanning a document");
    }

    @Override
    public void checkScanQuality() {
        System.out.println("Checking scan quality for " + modelName);
    }
}

public class Practical19 {
    public static void main(String[] args) {
        System.out.println("Demonstrating Multiple Inheritance Using Interfaces:\n");

        // Create an AllInOnePrinter object
        AllInOnePrinter allInOne = new AllInOnePrinter("HP OfficeJet Pro");

        // Create a BasicPrinter object
        BasicPrinter basicPrinter = new BasicPrinter("Canon ImageCLASS");

        // Demonstrate AllInOnePrinter functionality
        System.out.println("1. Testing AllInOnePrinter:");
        allInOne.print();
        allInOne.scan();
        allInOne.fax("123-456-7890");
        allInOne.displayStatus();

        // Demonstrate BasicPrinter functionality
        System.out.println("\n2. Testing BasicPrinter:");
        basicPrinter.print();
        basicPrinter.scan();
        basicPrinter.checkInk();
        basicPrinter.checkScanQuality();

        // Demonstrate polymorphism using interfaces
        System.out.println("\n3. Demonstrating polymorphism using interfaces:");

        // Using Printer interface
        System.out.println("\nTesting through Printer interface:");
        Printer printer = allInOne;
        printer.print();
        printer.checkInk();

        // Using Scanner interface
        System.out.println("\nTesting through Scanner interface:");
        Scanner scanner = allInOne;
        scanner.scan();
        scanner.checkScanQuality();

        // Using Fax interface
        System.out.println("\nTesting through Fax interface:");
        Fax fax = allInOne;
        fax.fax("987-654-3210");
        fax.checkFaxLine();
    }
}
```

Practical20

```java
// Practical20.java - Demonstrate abstract class and method overriding

// Abstract class Shape
abstract class Shape {
    protected String name;
    protected String color;

    // Constructor
    public Shape(String name, String color) {
        this.name = name;
        this.color = color;
    }

    // Abstract method to calculate area
    public abstract double area();

    // Concrete method to display shape info
    public void displayInfo() {
        System.out.println("Shape: " + name);
        System.out.println("Color: " + color);
        System.out.printf("Area: %.2f square units\n", area());
    }
}

// Triangle class
class Triangle extends Shape {
    private double base;
    private double height;

    public Triangle(String color, double base, double height) {
        super("Triangle", color);
        this.base = base;
        this.height = height;
    }

    @Override
    public double area() {
        return 0.5 * base * height;
    }

    @Override
    public void displayInfo() {
        super.displayInfo();
        System.out.println("Base: " + base + " units");
        System.out.println("Height: " + height + " units");
    }
}

// Rectangle class
class Rectangle extends Shape {
    private double length;
    private double width;

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

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

    @Override
    public void displayInfo() {
        super.displayInfo();
        System.out.println("Length: " + length + " units");
        System.out.println("Width: " + width + " units");
    }
}

// Circle class
class Circle extends Shape {
    private double radius;

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

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

    @Override
    public void displayInfo() {
        super.displayInfo();
        System.out.println("Radius: " + radius + " units");
    }
}

public class Practical20 {
    // Method to process any shape
    public static void processShape(Shape shape) {
        System.out.println("\nProcessing shape:");
```

```java
            shape.displayInfo();
        }

    public static void main(String[] args) {
        System.out.println("Demonstrating Abstract Class with Shape Hierarchy:\n");

        // Create instances of different shapes
        Triangle triangle = new Triangle("Red", 6.0, 4.0);
        Rectangle rectangle = new Rectangle("Blue", 5.0, 3.0);
        Circle circle = new Circle("Green", 3.0);

        // Process each shape using polymorphism
        processShape(triangle);
        processShape(rectangle);
        processShape(circle);

        // Demonstrate array of shapes
        System.out.println("\nProcessing array of shapes:");
        Shape[] shapes = {triangle, rectangle, circle};

        for (Shape shape : shapes) {
            System.out.println("\nShape Details:");
            shape.displayInfo();
            System.out.println("Calculated area: " + String.format("%.2f",
shape.area()));
        }

        // Demonstrate that we cannot instantiate abstract class
        // Following line would cause compilation error:
        // Shape shape = new Shape("Generic", "Yellow");

        // Calculate total area of all shapes
        double totalArea = 0;
        for (Shape shape : shapes) {
            totalArea += shape.area();
        }
        System.out.printf("\nTotal area of all shapes: %.2f square units\n", totalArea);
    }
}
```

Practical21

```java
// Practical21.java - Demonstrate use of final class

// Final class - cannot be inherited
final class SecureConfig {
    private String serverName;
    private String password;
    private int port;
    private boolean isSSLEnabled;
    // Constructor
    public SecureConfig(String serverName, String password, int port) {
        this.serverName = serverName;
        this.password = password;
        this.port = port;
        this.isSSLEnabled = true;
    }

    // Public methods to access and modify configuration
    public String getServerName() {
        return serverName;
    }

    public int getPort() {
        return port;
    }

    public boolean isSSLEnabled() {
        return isSSLEnabled;
    }

    public void setSSLEnabled(boolean enabled) {
        this.isSSLEnabled = enabled;
    }

    // Method to display configuration (excluding sensitive data)
    public void displayConfig() {
        System.out.println("Server Configuration:");
        System.out.println("Server Name: " + serverName);
        System.out.println("Port: " + port);
        System.out.println("SSL Enabled: " + isSSLEnabled);
        System.out.println("Password: *****"); // Hide actual password
    }

    // Method to validate configuration
    public boolean validateConfig() {
        return serverName != null && !serverName.isEmpty() &&
               password != null && !password.isEmpty() &&
               port > 0 && port <= 65535;
    }
}

// This class would cause compilation error if uncommented:
/*
class ExtendedConfig extends SecureConfig {  // Error: cannot inherit from final class
    private String additionalSetting;

    public ExtendedConfig(String serverName, String password, int port) {
        super(serverName, password, port);
    }
}
*/
```

```java
public class Practical21 {
    public static void main(String[] args) {
        System.out.println("Demonstrating Final Class Usage:\n");

        // Create instances of SecureConfig
        SecureConfig config1 = new SecureConfig("prod-server-1", "secretpass123", 443);
        SecureConfig config2 = new SecureConfig("dev-server-1", "devpass456", 8080);

        // Demonstrate config1
        System.out.println("1. First Configuration:");
        config1.displayConfig();
        System.out.println("Configuration Valid: " + config1.validateConfig());

        // Modify SSL settings
        config1.setSSLEnabled(false);
        System.out.println("\nAfter modifying SSL settings:");
        config1.displayConfig();

        // Demonstrate config2
        System.out.println("\n2. Second Configuration:");
        config2.displayConfig();
        System.out.println("Configuration Valid: " + config2.validateConfig());

        // Demonstrate accessing individual properties
        System.out.println("\n3. Accessing Individual Properties:");
        System.out.println("Server Name: " + config2.getServerName());
        System.out.println("Port: " + config2.getPort());
        System.out.println("SSL Enabled: " + config2.isSSLEnabled());

        // Create array of configurations
        System.out.println("\n4. Processing Multiple Configurations:");
        SecureConfig[] configs = {config1, config2};

        for (int i = 0; i < configs.length; i++) {
            System.out.println("\nConfiguration " + (i + 1) + ":");
            configs[i].displayConfig();
        }
    }
}
```

Practical22

```java
// File: Practical22.java
import shapes.Circle;
import shapes.Rectangle;
import util.Calculator;

public class Practical22 {
    public static void main(String[] args) {
        System.out.println("Demonstrating Package Usage:\n");
        // Create objects of classes from shapes package
        Circle circle = new Circle(5.0);
        Rectangle rectangle = new Rectangle(4.0, 6.0);

        // Calculate and display circle measurements
        System.out.println("Circle Measurements:");
        System.out.println("Area: " +
            Calculator.round(circle.getArea(), 2) + " square units");
        System.out.println("Perimeter: " +
            Calculator.round(circle.getPerimeter(), 2) + " units");

        // Calculate and display rectangle measurements
        System.out.println("\nRectangle Measurements:");
        System.out.println("Area: " +
            Calculator.round(rectangle.getArea(), 2) + " square units");
        System.out.println("Perimeter: " +
            Calculator.round(rectangle.getPerimeter(), 2) + " units");

        // Demonstrate fully qualified names
        System.out.println("\nUsing fully qualified names:");
        shapes.Circle circle2 = new shapes.Circle(3.0);
        System.out.println("New circle area: " +
            util.Calculator.round(circle2.getArea(), 2) + " square units");
    }
}
```

```java
// File: shapes/Circle.java
package shapes;

public class Circle {
    private double radius;

    public Circle(double radius) {
        this.radius = radius;
    }

    public double getArea() {
        return Math.PI * radius * radius;
    }

    public double getPerimeter() {
        return 2 * Math.PI * radius;
    }
}
```

```java
// File: shapes/Rectangle.java
package shapes;

public class Rectangle {
    private double length;
    private double width;

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

    public double getArea() {
        return length * width;
    }

    public double getPerimeter() {
        return 2 * (length + width);
    }
}
```

```java
// File: util/Calculator.java
package util;

public class Calculator {
    public static double round(double value, int places) {
        double scale = Math.pow(10, places);
        return Math.round(value * scale) / scale;
    }
}
```

Practical23

```java
// Practical23.java - Demonstrate user defined exception for divide by zero

// Custom Exception class
class DivideByZeroException extends Exception {
    public DivideByZeroException() {
        super("Cannot divide by zero!");
    }

    public DivideByZeroException(String message) {
        super(message);
    }
}

// Calculator class with division method
class Calculator {
    // Method that throws our custom exception
    public static double divide(double numerator, double denominator)
            throws DivideByZeroException {
        if (denominator == 0) {
            throw new DivideByZeroException(
                "Division by zero error! Numerator was: " + numerator);
        }
        return numerator / denominator;
    }
}

public class Practical23 {
    // Method to demonstrate division with exception handling
    public static void performDivision(double numerator, double denominator) {
        try {
            double result = Calculator.divide(numerator, denominator);
            System.out.printf("%.2f ÷ %.2f = %.2f\n", numerator, denominator, result);
        } catch (DivideByZeroException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }

    public static void main(String[] args) {
        System.out.println("Demonstrating User Defined Exception:\n");

        // Test cases
        System.out.println("1. Normal division:");
        performDivision(10.0, 2.0);

        System.out.println("\n2. Division by zero:");
        performDivision(20.0, 0.0);

        System.out.println("\n3. Multiple divisions in a loop:");
        double[] numerators = {15.0, 25.0, 30.0};
        double[] denominators = {3.0, 0.0, 5.0};

        for (int i = 0; i < numerators.length; i++) {
            System.out.println("\nAttempting division " + (i + 1) + ":");
            performDivision(numerators[i], denominators[i]);
        }

        // Demonstrating exception handling with try-catch block
        System.out.println("\n4. Direct try-catch usage:");
        try {
            System.out.println("Attempting risky division...");
            double result = Calculator.divide(50.0, 0.0);
            System.out.println("Result of division: " + result); // This line won't be
reached
        } catch (DivideByZeroException e) {
            System.out.println("Caught exception: " + e.getMessage());
        } finally {
            System.out.println("Finally block executed");
        }

        // Additional demonstration with successful division
        System.out.println("\n5. Another try-catch example with successful division:");
        try {
            System.out.println("Attempting safe division...");
            double result = Calculator.divide(50.0, 2.0);
            System.out.println("Result of division: " + result); // This line will be
reached
        } catch (DivideByZeroException e) {
            System.out.println("Caught exception: " + e.getMessage());
        } finally {
            System.out.println("Finally block executed");
        }

        System.out.println("\n6. Program continues after exception handling");
        performDivision(100.0, 25.0);
    }
}
```

Practical24

```
// Practical24.java - Banking Application with custom exception

// Custom Exception for insufficient funds
class InsufficientFundsException extends Exception {
    private double currentBalance;
    private double withdrawAmount;

    public InsufficientFundsException(double currentBalance, double withdrawAmount) {
        super(String.format("Not Sufficient Fund! Balance: $%.2f, Withdrawal Amount:
$%.2f",
              currentBalance, withdrawAmount));
        this.currentBalance = currentBalance;
        this.withdrawAmount = withdrawAmount;
    }

    public double getDeficit() {
        return withdrawAmount - currentBalance;
    }
}

// Bank Account class
class BankAccount {
    private double balance;
    private String accountNumber;

    public BankAccount(String accountNumber, double initialDeposit) {
        this.accountNumber = accountNumber;
        this.balance = initialDeposit;
    }

    public void deposit(double amount) {
        if (amount > 0) {
            balance += amount;
            System.out.printf("Deposited: $%.2f\n", amount);
            displayBalance();
        } else {
            System.out.println("Invalid deposit amount");
        }
    }

    public void withdraw(double amount) throws InsufficientFundsException {
        if (amount > balance) {
            throw new InsufficientFundsException(balance, amount);
        }

        balance -= amount;
        System.out.printf("Withdrawn: $%.2f\n", amount);
        displayBalance();
    }

    public void displayBalance() {
        System.out.printf("Current Balance: $%.2f\n", balance);
    }

    public double getBalance() {
        return balance;
    }

    public String getAccountNumber() {
        return accountNumber;
    }
}

public class Practical24 {
    public static void main(String[] args) {
        System.out.println("Banking Application Demonstration:\n");

        // Create a bank account with initial deposit of $25000
        BankAccount account = new BankAccount("ACC001", 25000);
        System.out.println("Account created successfully!");
        account.displayBalance();

        try {
            // Performing transactions as per requirement
            System.out.println("\n1. Withdrawing $20000:");
            account.withdraw(20000);

            System.out.println("\n2. Withdrawing $4000:");
            account.withdraw(4000);

            System.out.println("\n3. Attempting to withdraw $2000:");
            account.withdraw(2000);

        } catch (InsufficientFundsException e) {
            System.out.println("Transaction Failed: " + e.getMessage());
            System.out.printf("Deficit Amount: $%.2f\n", e.getDeficit());
        }

        // Additional demonstrations
        System.out.println("\nAdditional Operations:");

        try {
            // Deposit some money
            System.out.println("\n4. Depositing $1000:");
            account.deposit(1000);

            // Try withdrawal again
            System.out.println("\n5. Attempting to withdraw $1500:");
            account.withdraw(1500);

        } catch (InsufficientFundsException e) {
            System.out.println("Transaction Failed: " + e.getMessage());
            System.out.printf("Deficit Amount: $%.2f\n", e.getDeficit());
        }

        // Final balance check
        System.out.println("\nFinal Account Status:");
        account.displayBalance();
    }
}
```

Practical25

```
// Practical25.java - Demonstrate thread creation and execution

// First thread class
class Thread1 extends Thread {
    @Override
    public void run() {
        try {
            for (int i = 1; i <= 5; i++) {
                System.out.println("Thread1");
                // Sleep for 1000 milliseconds (1 second)
                Thread.sleep(1000);
            }
        } catch (InterruptedException e) {
            System.out.println("Thread1 interrupted");
        }
    }
}

// Second thread class
class Thread2 extends Thread {
    @Override
    public void run() {
        try {
```

```java
            for (int i = 1; i <= 5; i++) {
                System.out.println("Thread2");
                // Sleep for 2000 milliseconds (2 seconds)
                Thread.sleep(2000);
            }
        } catch (InterruptedException e) {
            System.out.println("Thread2 interrupted");
        }
    }
}

public class Practical25 {
    public static void main(String[] args) {
        System.out.println("Demonstrating Thread Creation and Execution:\n");

        // Create thread objects
        Thread1 t1 = new Thread1();
        Thread2 t2 = new Thread2();

        // Set thread names
        t1.setName("Thread-1");
        t2.setName("Thread-2");

        // Display thread information before starting
        System.out.println("Thread States Before Starting:");
        System.out.println(t1.getName() + " State: " + t1.getState());
        System.out.println(t2.getName() + " State: " + t2.getState());

        System.out.println("\nStarting threads...");

        // Start both threads
        t1.start();
        t2.start();

        // Display thread information after starting
        System.out.println("\nThread States After Starting:");
        System.out.println(t1.getName() + " State: " + t1.getState());
        System.out.println(t2.getName() + " State: " + t2.getState());

        // Wait for both threads to complete
        try {
            t1.join();
            t2.join();
        } catch (InterruptedException e) {
            System.out.println("Main thread interrupted");
        }

        // Display final thread states
        System.out.println("\nThread States After Completion:");
        System.out.println(t1.getName() + " State: " + t1.getState());
        System.out.println(t2.getName() + " State: " + t2.getState());
        System.out.println("\nMain thread ending");
    }
}
```

Practical26

```
// Practical26.java - Demonstrate threads printing even and odd numbers

class NumberPrinter {
    private int currentNumber = 1;
    private final int maxNumber = 200;
    private boolean isEvenTurn = false;

    // Method for printing even numbers
    synchronized void printEven() {
        while (currentNumber <= maxNumber) {
            try {
                // Wait if it's not even number's turn
                while (!isEvenTurn && currentNumber <= maxNumber) {
                    wait();
                }

                if (currentNumber <= maxNumber) {
                    System.out.printf("Even Thread: %d%n", currentNumber);
                    currentNumber++;
                    isEvenTurn = false;
                    notify();
                }
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }
        }
    }

    // Method for printing odd numbers
    synchronized void printOdd() {
        while (currentNumber <= maxNumber) {
            try {
                // Wait if it's not odd number's turn
                while (isEvenTurn && currentNumber <= maxNumber) {
                    wait();
                }

                if (currentNumber <= maxNumber) {
                    System.out.printf("Odd Thread: %d%n", currentNumber);
                    currentNumber++;
                    isEvenTurn = true;
                    notify();
                }
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }
        }
    }
}

// Thread class for printing even numbers
class EvenPrinter extends Thread {
    private NumberPrinter printer;

    public EvenPrinter(NumberPrinter printer) {
        this.printer = printer;
    }

    @Override
    public void run() {
        printer.printEven();
    }
}

// Thread class for printing odd numbers
class OddPrinter extends Thread {
    private NumberPrinter printer;

    public OddPrinter(NumberPrinter printer) {
        this.printer = printer;
    }

    @Override
    public void run() {
        printer.printOdd();
    }
}

public class Practical26 {
    public static void main(String[] args) {
        System.out.println("Demonstrating Even-Odd Number Printing Using Threads:\n");

        // Create shared number printer object
        NumberPrinter printer = new NumberPrinter();

        // Create even and odd printer threads
        Thread evenThread = new EvenPrinter(printer);
        Thread oddThread = new OddPrinter(printer);

        // Set thread names
        evenThread.setName("EvenThread");
        oddThread.setName("OddThread");

        // Start both threads
        System.out.println("Starting threads to print numbers from 1 to 200...\n");
        oddThread.start();
        evenThread.start();
        // Wait for both threads to complete
        try {
            evenThread.join();
            oddThread.join();
        } catch (InterruptedException e) {
            System.out.println("Main thread interrupted");
        }

        System.out.println("\nBoth threads completed execution");
    }
}
```

Practical27

```
// Practical27.java - Demonstrate read and write operations on a text file

import java.io.*;
import java.util.Scanner;

public class Practical27 {
    // Method to write content to a file
    public static void writeToFile(String fileName, String content) {
        try (FileWriter writer = new FileWriter(fileName);
             BufferedWriter bufferedWriter = new BufferedWriter(writer)) {

            bufferedWriter.write(content);
            System.out.println("Successfully wrote to the file.");

        } catch (IOException e) {
            System.out.println("An error occurred while writing to the file:");
            e.printStackTrace();
        }
    }

    // Method to append content to a file
    public static void appendToFile(String fileName, String content) {
        try (FileWriter writer = new FileWriter(fileName, true);
             BufferedWriter bufferedWriter = new BufferedWriter(writer)) {

            bufferedWriter.write(content);
            System.out.println("Successfully appended to the file.");

        } catch (IOException e) {
            System.out.println("An error occurred while appending to the file:");
            e.printStackTrace();
        }
    }

    // Method to read content from a file using BufferedReader
    public static void readFileUsingBufferedReader(String fileName) {
        try (FileReader reader = new FileReader(fileName);
             BufferedReader bufferedReader = new BufferedReader(reader)) {

            System.out.println("\nReading file using BufferedReader:");
            String line;
            while ((line = bufferedReader.readLine()) != null) {
                System.out.println(line);
            }

        } catch (IOException e) {
            System.out.println("An error occurred while reading the file:");
            e.printStackTrace();
        }
    }

    // Method to read content from a file using Scanner
    public static void readFileUsingScanner(String fileName) {
        try (Scanner scanner = new Scanner(new File(fileName))) {
            System.out.println("\nReading file using Scanner:");
            while (scanner.hasNextLine()) {
                System.out.println(scanner.nextLine());
            }

        } catch (FileNotFoundException e) {
            System.out.println("File not found:");
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
        System.out.println("Demonstrating File Operations:\n");

        String fileName = "sample.txt";

        // Write initial content to file
        System.out.println("1. Writing initial content to file:");
        String initialContent = "Hello! This is line 1.\n" +
                                "This is line 2.\n" +
                                "This is line 3.\n";
        writeToFile(fileName, initialContent);

        // Read the file content using BufferedReader
        readFileUsingBufferedReader(fileName);

        // Append additional content
        System.out.println("\n2. Appending content to file:");
        String additionalContent = "This is line 4 (appended).\n" +
                                   "This is line 5 (appended).\n";
        appendToFile(fileName, additionalContent);

        // Read the file content using Scanner
        readFileUsingScanner(fileName);
```

```java
        // Demonstrate File class operations
        File file = new File(fileName);
        System.out.println("\n3. File Information:");
        System.out.println("File exists: " + file.exists());
        System.out.println("File name: " + file.getName());
        System.out.println("Absolute path: " + file.getAbsolutePath());
        System.out.println("File size: " + file.length() + " bytes");
        System.out.println("Can read: " + file.canRead());
        System.out.println("Can write: " + file.canWrite());

        // Clean up - Delete the file
        System.out.println("\n4. Cleaning up:");
        if (file.delete()) {
            System.out.println("File deleted successfully.");
        } else {
            System.out.println("Failed to delete the file.");
        }
    }
}
```

Practical28

```java
// Practical28.java - Demonstrate use of List (ArrayList and LinkedList)

import java.util.*;

public class Practical28 {
    // Method to display list contents
    public static void displayList(List<?> list, String listName) {
        System.out.println(listName + " contents:");
        for (Object item : list) {
            System.out.print(item + " ");
        }
        System.out.println("\nSize: " + list.size());
        System.out.println();
    }

    public static void main(String[] args) {
        System.out.println("Demonstrating List Interface Usage:\n");

        // Create ArrayList and add weekdays
        List<String> weekdays = new ArrayList<>();
        System.out.println("1. ArrayList Operations (Weekdays):");

        // Adding weekdays
        weekdays.add("Monday");
        weekdays.add("Tuesday");
        weekdays.add("Wednesday");
        weekdays.add("Thursday");
        weekdays.add("Friday");
        displayList(weekdays, "Weekdays ArrayList");

        // Demonstrate ArrayList operations
        System.out.println("ArrayList Operations:");
        System.out.println("First day: " + weekdays.get(0));
        System.out.println("Last day: " + weekdays.get(weekdays.size() - 1));
        System.out.println("Contains 'Wednesday'? " + weekdays.contains("Wednesday"));
        System.out.println("Index of 'Friday': " + weekdays.indexOf("Friday"));

        // Create LinkedList and add months
        List<String> months = new LinkedList<>();
        System.out.println("\n2. LinkedList Operations (Months):");

        // Adding months
        months.add("January");
        months.add("February");
        months.add("March");
        months.add("April");
        months.add("May");
        months.add("June");

        displayList(months, "Months LinkedList");

        // Demonstrate LinkedList specific operations
        LinkedList<String> monthsLinked = (LinkedList<String>) months;
        System.out.println("LinkedList Specific Operations:");
        System.out.println("First month: " + monthsLinked.getFirst());
        System.out.println("Last month: " + monthsLinked.getLast());

        // Add elements at specific positions
        monthsLinked.addFirst("December");  // Add at beginning
        monthsLinked.addLast("July");      // Add at end

        System.out.println("\nAfter adding elements:");
        displayList(months, "Updated Months LinkedList");

        // Demonstrate List interface common operations
        System.out.println("3. Common List Operations:");

        // Sorting
        Collections.sort(weekdays);
        System.out.println("Sorted weekdays:");
        displayList(weekdays, "Sorted Weekdays");

        Collections.sort(months);
        System.out.println("Sorted months:");
        displayList(months, "Sorted Months");

        // Removing elements
        weekdays.remove("Friday");
        months.remove("July");
        System.out.println("After removing elements:");
        System.out.println("Weekdays after removing Friday:");
        displayList(weekdays, "Modified Weekdays");
        System.out.println("Months after removing July:");
        displayList(months, "Modified Months");

        // Clear lists
        weekdays.clear();
        months.clear();

        System.out.println("4. After clearing lists:");
        displayList(weekdays, "Cleared Weekdays");
        displayList(months, "Cleared Months");
    }
}
```

Practical29

```
// Practical29.java - Demonstrate HashSet operations with colors

import java.util.HashSet;
import java.util.Iterator;

public class Practical29 {
    // Method to display set contents with iteration count
    private static void displaySet(HashSet<String> set, String message) {
        System.out.println(message);
        int count = 1;
        for (String color : set) {
            System.out.println(count + ". " + color);
            count++;
        }
        System.out.println("Set size: " + set.size() + "\n");
    }

    public static void main(String[] args) {
        System.out.println("Demonstrating HashSet Operations with Colors:\n");

        // Create a HashSet to store colors
        HashSet<String> colors = new HashSet<>();

        // 1. Adding colors to the set
        System.out.println("1. Adding colors to HashSet:");
        colors.add("Red");
        colors.add("Green");
        colors.add("Blue");
        colors.add("Yellow");
        colors.add("Purple");

        displaySet(colors, "Initial Set of Colors:");
```

```java
        // 2. Demonstrate duplicate handling
        System.out.println("2. Attempting to add duplicate colors:");
        boolean addedRed = colors.add("Red");
        boolean addedOrange = colors.add("Orange");

        System.out.println("Added 'Red' again? " + addedRed);
        System.out.println("Added 'Orange'? " + addedOrange);
        displaySet(colors, "Set after attempting duplicates:");

        // 3. Different ways of iteration
        System.out.println("3. Different iteration methods:");

        // Using Iterator
        System.out.println("Using Iterator:");
        Iterator<String> iterator = colors.iterator();
        while (iterator.hasNext()) {
            System.out.println(iterator.next());
        }
        System.out.println();

        // Using forEach method
        System.out.println("Using forEach method:");
        colors.forEach(color -> System.out.println(color));
        System.out.println();

        // 4. Searching and removing elements
        System.out.println("4. Search and remove operations:");
        System.out.println("Contains 'Blue'? " + colors.contains("Blue"));
        System.out.println("Contains 'Black'? " + colors.contains("Black"));

        // Remove a color
        boolean removed = colors.remove("Yellow");
        System.out.println("Removed 'Yellow'? " + removed);
        displaySet(colors, "Set after removing 'Yellow':");

        // 5. Create a new set for set operations
        HashSet<String> moreColors = new HashSet<>();
        moreColors.add("Pink");
        moreColors.add("Blue");  // Duplicate with first set
        moreColors.add("Brown");

        System.out.println("5. Set operations with new colors:");
        displaySet(moreColors, "New set of colors:");

        // Add all elements from moreColors to colors
        colors.addAll(moreColors);
        displaySet(colors, "After adding all new colors:");

        // 6. Clear the set
        System.out.println("6. Clearing the set:");
        colors.clear();
        System.out.println("Is set empty? " + colors.isEmpty());
        displaySet(colors, "Set after clearing:");
    }
}
```

Practical30

```
// Practical30.java - Demonstrate HashMap with Student Data

import java.util.HashMap;
import java.util.Map;

public class Practical30 {
    // Method to display student data
    private static void displayStudents(HashMap<String, String> students, String message)
{
        System.out.println(message);
        if (students.isEmpty()) {
            System.out.println("No students in the map.");
        } else {
            for (Map.Entry<String, String> entry : students.entrySet()) {
                System.out.printf("Enrollment No: %s, Name: %s%n",
                    entry.getKey(), entry.getValue());
            }
        }
        System.out.println("Total students: " + students.size() + "\n");
    }

    public static void main(String[] args) {
        System.out.println("Demonstrating HashMap Operations with Student Data:\n");

        // Create HashMap to store student data
        HashMap<String, String> students = new HashMap<>();

        // 1. Adding students to the map
        System.out.println("1. Adding students to HashMap:");
        students.put("A101", "John Smith");
        students.put("A102", "Emma Watson");
        students.put("A103", "Michael Johnson");
        students.put("A104", "Sarah Wilson");
        students.put("A105", "David Brown");

        displayStudents(students, "Initial Student List:");

        // 2. Accessing specific student
        System.out.println("2. Accessing student data:");
        String enrollmentNo = "A103";
        System.out.println("Student with enrollment no " + enrollmentNo + ": "
            + students.get(enrollmentNo));

        // Try accessing non-existent student
        System.out.println("Student with enrollment no A106: "
            + students.get("A106"));
        System.out.println();

        // 3. Updating student data
        System.out.println("3. Updating student data:");
        students.put("A101", "John Smith Jr.");  // Update existing entry
        System.out.println("Updated A101's name");
        displayStudents(students, "After updating:");

        // 4. Checking existence
        System.out.println("4. Checking existence:");
        System.out.println("Contains enrollment no A102? "
            + students.containsKey("A102"));
        System.out.println("Contains student Emma Watson? "
            + students.containsValue("Emma Watson"));
        System.out.println();

        // 5. Different ways to iterate
        System.out.println("5. Different ways to iterate through the map:");

        // Using entrySet
        System.out.println("Using entrySet:");
        for (Map.Entry<String, String> entry : students.entrySet()) {
            System.out.println("Key: " + entry.getKey() + ", Value: " +
entry.getValue());
        }
        System.out.println();

        // Using keySet
        System.out.println("Using keySet:");
        for (String key : students.keySet()) {
            System.out.println("Enrollment No: " + key);
        }
        System.out.println();

        // Using values
        System.out.println("Using values:");
        for (String value : students.values()) {
            System.out.println("Student Name: " + value);
        }
        System.out.println();

        // 6. Removing a student
        System.out.println("6. Removing student:");
        String removedStudent = students.remove("A104");
        System.out.println("Removed student: " + removedStudent);
        displayStudents(students, "After removing A104:");

        // 7. Using getOrDefault
        System.out.println("7. Using getOrDefault:");
        System.out.println("Student A105: " +
            students.getOrDefault("A105", "Not Found"));
        System.out.println("Student A106: " +
            students.getOrDefault("A106", "Not Found"));
        System.out.println();

        // 8. Clear the map
        System.out.println("8. Clearing the map:");
        students.clear();
        System.out.println("Is map empty? " + students.isEmpty());
        displayStudents(students, "After clearing:");
    }
}
```