EXPERIMENT 1

```
import java.util.Scanner;

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

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

        // Validate input
        if (count <= 0) {
            System.out.println("Number of elements must be positive.");
            return;
        }

        double sum = 0;

        System.out.println("Enter " + count + " numbers:");
        for (int i = 0; i < count; i++) {
            System.out.print("Number " + (i + 1) + ": ");
            double number = scanner.nextDouble();
            sum += number;
        }

        double average = sum / count;

        System.out.println("\nResults:");
        System.out.println("Sum: " + sum);
        System.out.println("Average: " + average);

        scanner.close();
    }
}
```

EXPERIMENT 2

```
import java.util.Scanner;
public class Calculator {
public static void main(String[] args) {
Scanner scanner = new Scanner(System.in);

        // Input numbers and operator
        System.out.print("Enter the first number: ");
        double num1 = scanner.nextDouble();

        System.out.print("Enter the operator (+, -, *, /): ");
        char op = scanner.next().charAt(0);

        System.out.print("Enter the second number: ");
        double num2 = scanner.nextDouble();
```

```
        double result;

        // Switch-case for calculator operations
        switch (op) {
            case '+':
                result = num1 + num2;
                System.out.println("Result: " + result);
                break;

            case '-':
                result = num1 - num2;
                System.out.println("Result: " + result);
                break;

            case '*':
                result = num1 * num2;
                System.out.println("Result: " + result);
                break;

            case '/':
                if (num2 != 0) {
                    result = num1 / num2;
                    System.out.println("Result: " + result);
                } else {
                    System.out.println("Error: Division by zero is not allowed.");
                }
                break;

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

        scanner.close();
    }
}
```

EXPERIMENT 3

```
import java.util.Scanner;

// Complete working program with Student class and main method
public class StudentProgram {
    // Student class definition
    static class Student {
        private String name;
        private int rollNumber;
        private double marks;

        public Student(String name, int rollNumber, double marks) {
            this.name = name;
            this.rollNumber = rollNumber;
            this.marks = marks;
        }
```

```
        public String getName() { return name; }
        public int getRollNumber() { return rollNumber; }
        public double getMarks() { return marks; }

        public void setName(String name) { this.name = name; }
        public void setRollNumber(int rollNumber) { this.rollNumber = rollNumber; }
        public void setMarks(double marks) { this.marks = marks; }

        public void displayInfo() {
            System.out.println("\nStudent Information:");
            System.out.println("Name: " + name);
            System.out.println("Roll Number: " + rollNumber);
            System.out.println("Marks: " + marks);
        }
    }

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

        System.out.println("Student Information System");
        System.out.println("--------------------------");

        // Input student details
        System.out.print("Enter student name: ");
        String name = scanner.nextLine();

        System.out.print("Enter roll number: ");
        int rollNumber = scanner.nextInt();

        System.out.print("Enter marks: ");
        double marks = scanner.nextDouble();

        // Create student object
        Student student = new Student(name, rollNumber, marks);

        // Display student info
        student.displayInfo();

        // Option to update marks
        System.out.print("\nDo you want to update marks? (y/n): ");
        char choice = scanner.next().charAt(0);

        if (choice == 'y' || choice == 'Y') {
            System.out.print("Enter new marks: ");
            double newMarks = scanner.nextDouble();
            student.setMarks(newMarks);
            System.out.println("Marks updated successfully!");
            student.displayInfo();
        }

        System.out.println("\nThank you for using the Student Information
System!");
        scanner.close();
    }
}
```

EXPERIMENT 4

```
public class OverloadThisDemo {
    private String name;
    private int value;

    // Constructor overloading with 'this'
    public OverloadThisDemo() {
        this("Default", 0); // Calls 2-arg constructor
    }

    public OverloadThisDemo(String name, int value) {
        this.name = name;   // 'this' distinguishes parameter from field
        this.value = value;
    }

    // Method overloading examples
    public void show() {
        System.out.println(name + ": " + value);
    }

    public void show(String prefix) { // Overloaded version
        System.out.println(prefix + " " + name + ": " + value);
    }

    // Method using 'this' to call another method
    public void display() {
        this.show("[Display]"); // Calls overloaded method
    }

    public static void main(String[] args) {
        OverloadThisDemo obj1 = new OverloadThisDemo();
        OverloadThisDemo obj2 = new OverloadThisDemo("Test", 42);

        obj1.show();        // Calls simple show()
        obj2.display();    // Calls show() via display()
    }
}
```

EXPERIMENT 5

```
// Complete runnable program demonstrating inheritance
public class InheritanceDemo {

    // Single Inheritance
    static class Animal {
        void eat() {
            System.out.println("Animal is eating");
        }
    }
```

```
    static class Dog extends Animal {  // Single inheritance (Animal → Dog)
        void bark() {
            System.out.println("Dog is barking");
        }
    }

    // Multilevel Inheritance
    static class Puppy extends Dog {  // Multilevel (Animal → Dog → Puppy)
        void weep() {
            System.out.println("Puppy is weeping");
        }
    }

    public static void main(String[] args) {
        System.out.println("=== Single Inheritance Demo ===");
        Dog myDog = new Dog();
        myDog.eat();  // Inherited from Animal
        myDog.bark(); // Own method

        System.out.println("\n=== Multilevel Inheritance Demo ===");
        Puppy myPuppy = new Puppy();
        myPuppy.eat();   // From Animal
        myPuppy.bark();  // From Dog
        myPuppy.weep();  // Own method
    }
}
```

EXPERIMENT 6

```
// Shape.java
interface Shape {
double calculateArea();
}
// Circle.java
class Circle implements Shape {
private double radius;
public Circle(double radius) {
this.radius = radius;
}
@Override
public double calculateArea() {
return Math.PI * radius * radius;
}
}
// Rectangle.java
class Rectangle implements Shape {
private double length;
private double width;
public Rectangle(double length, double width) {
this.length = length;
this.width = width;
}
```

```
@Override
public double calculateArea() {
return length * width;
}
}
// Triangle.java
class Triangle implements Shape {
private double base;
private double height;
public Triangle(double base, double height) {
this.base = base;
this.height = height;
}
@Override
public double calculateArea() {
return 0.5 * base * height;
}
}
// Main.java
public class Main {
public static void main(String[] args) {
Shape circle = new Circle(5);
Shape rectangle = new Rectangle(4, 6);
Shape triangle = new Triangle(3, 8);
System.out.println("Area of Circle: " + circle.calculateArea());
System.out.println("Area of Rectangle: " + rectangle.calculateArea());
System.out.println("Area of Triangle: " + triangle.calculateArea());
}
}
```

EXPERIMENT 7

```
import java.util.InputMismatchException;
import java.util.Scanner;

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

        try {
            System.out.print("Enter numerator: ");
            int numerator = scanner.nextInt();

            System.out.print("Enter denominator: ");
            int denominator = scanner.nextInt();

            int result = divideNumbers(numerator, denominator);
            System.out.println("Result: " + result);

        } catch (ArithmeticException e) {
            System.out.println("Error: Division by zero is not allowed!");
        } catch (InputMismatchException e) {
            System.out.println("Error: Please enter valid integers only!");
```

```
        } finally {
            scanner.close();
            System.out.println("Program execution completed.");
        }
    }

    public static int divideNumbers(int a, int b) throws ArithmeticException {
        if (b == 0) {
            throw new ArithmeticException();
        }
        return a / b;
    }
}
```

EXPERIMENT 8

```
import java.util.Scanner;
import java.util.InputMismatchException;

public class BankingSystem {
    private static double balance = 1000.00; // Initial balance

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

        System.out.println("=== Simple Banking System ===");
        System.out.printf("Initial Balance: $%.2f\n", balance);

        while (running) {
            System.out.println("\n1. Deposit");
            System.out.println("2. Withdraw");
            System.out.println("3. Check Balance");
            System.out.println("4. Exit");
            System.out.print("Enter your choice: ");

            try {
                int choice = scanner.nextInt();

                switch (choice) {
                    case 1:
                        deposit(scanner);
                        break;
                    case 2:
                        withdraw(scanner);
                        break;
                    case 3:
                        checkBalance();
                        break;
                    case 4:
                        running = false;
                        System.out.println("Thank you for banking with us!");
```

```
                        break;
                    default:
                        System.out.println("Invalid choice! Please try again.");
                }
            } catch (InputMismatchException e) {
                System.out.println("Error: Please enter a valid number!");
                scanner.next(); // Clear invalid input
            } catch (InsufficientFundsException e) {
                System.out.println("Error: " + e.getMessage());
            } catch (InvalidAmountException e) {
                System.out.println("Error: " + e.getMessage());
            }
        }
        scanner.close();
    }

    private static void deposit(Scanner scanner) throws InvalidAmountException {
        System.out.print("Enter deposit amount: $");
        double amount = scanner.nextDouble();

        if (amount <= 0) {
            throw new InvalidAmountException("Deposit amount must be positive");
        }

        balance += amount;
        System.out.printf("$%.2f deposited successfully!\n", amount);
        checkBalance();
    }

    private static void withdraw(Scanner scanner) throws
InsufficientFundsException, InvalidAmountException {
        System.out.print("Enter withdrawal amount: $");
        double amount = scanner.nextDouble();

        if (amount <= 0) {
            throw new InvalidAmountException("Withdrawal amount must be positive");
        }

        if (amount > balance) {
            throw new InsufficientFundsException("Insufficient funds for
withdrawal");
        }

        balance -= amount;
        System.out.printf("$%.2f withdrawn successfully!\n", amount);
        checkBalance();
    }

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

// Custom Exceptions
class InsufficientFundsException extends Exception {
    public InsufficientFundsException(String message) {
        super(message);
    }
}
```

```
class InvalidAmountException extends Exception {
    public InvalidAmountException(String message) {
        super(message);
    }
}
```

EXPERIMENT 9

```
public class ConsoleWelcome {
    public static void main(String[] args) throws InterruptedException {
        // ANSI color codes for colored text output
        final String[] COLORS = {
            "\u001B[41m", // Red background
            "\u001B[42m", // Green background
            "\u001B[44m", // Blue background
            "\u001B[43m", // Yellow background
            "\u001B[45m", // Purple background
            "\u001B[46m"  // Cyan background
        };

        final String RESET = "\u001B[0m";

        while (true) {
            // Clear console (works in most terminals)
            System.out.print("\033[H\033[2J");
            System.out.flush();

            // Select random color
            String color = COLORS[(int)(Math.random() * COLORS.length)];

            // Display welcome message
            System.out.println(color + "====================" + RESET);
            System.out.println(color + "|                  |" + RESET);
            System.out.println(color + "|   WELCOME TO JAVA  |" + RESET);
            System.out.println(color + "|                  |" + RESET);
            System.out.println(color + "====================" + RESET);

            // Show current color name
            switch(color) {
                case "\u001B[41m": System.out.println("Current Color: Red"); break;
                case "\u001B[42m": System.out.println("Current Color: Green");
break;
                case "\u001B[44m": System.out.println("Current Color: Blue");
break;
                case "\u001B[43m": System.out.println("Current Color: Yellow");
break;
                case "\u001B[45m": System.out.println("Current Color: Purple");
break;
                case "\u001B[46m": System.out.println("Current Color: Cyan");
break;
            }
```

```
            // Wait 2 seconds before changing
            Thread.sleep(2000);
        }
    }
}
```

EXPERIMENT 10

```
import java.util.Scanner;

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

        System.out.println("Simple Text Calculator");
        System.out.println("Available operations: +, -, *, /");
        System.out.println("Enter 'exit' to quit");

        while (true) {
            System.out.print("\nEnter first number: ");
            String input1 = scanner.nextLine();
            if (input1.equalsIgnoreCase("exit")) break;

            System.out.print("Enter operator: ");
            String op = scanner.nextLine();
            if (op.equalsIgnoreCase("exit")) break;

            System.out.print("Enter second number: ");
            String input2 = scanner.nextLine();
            if (input2.equalsIgnoreCase("exit")) break;

            try {
                double num1 = Double.parseDouble(input1);
                double num2 = Double.parseDouble(input2);
                double result = 0;

                switch (op) {
                    case "+": result = num1 + num2; break;
                    case "-": result = num1 - num2; break;
                    case "*": result = num1 * num2; break;
                    case "/":
                        if (num2 == 0) {
                            System.out.println("Error: Cannot divide by zero!");
                            continue;
                        }
                        result = num1 / num2;
                        break;
                    default:
                        System.out.println("Invalid operator!");
                        continue;
                }

                System.out.printf("Result: %.2f %s %.2f = %.2f\n", num1, op, num2,
result);
```

```
            } catch (NumberFormatException e) {
                System.out.println("Error: Please enter valid numbers!");
            }
        }

        System.out.println("Calculator closed");
        scanner.close();
    }
}
```

EXPERIMENT 11

```
/**
 * Singleton Design Pattern Demonstration
 * Real-world use case: Application Configuration Manager
 */
public class SingletonDemo {
    public static void main(String[] args) {
        // Get the configuration manager instance
        ConfigManager configManager = ConfigManager.getInstance();

        // Set some configuration values
        configManager.set("database.url", "jdbc:mysql://localhost:3306/mydb");
        configManager.set("database.user", "admin");
        configManager.set("app.timeout", "5000");

        // Get the same instance again
        ConfigManager anotherReference = ConfigManager.getInstance();

        // Verify it's the same instance
        System.out.println("Same instance? " + (configManager ==
anotherReference));

        // Access configuration values
        System.out.println("Database URL: " + configManager.get("database.url"));
        System.out.println("Timeout: " + configManager.get("app.timeout"));
    }
}

/**
 * Singleton Configuration Manager
 * Real-world use: Centralized configuration management for an application
 */
class ConfigManager {
    // 1. Private static instance (volatile for thread safety)
    private static volatile ConfigManager instance;

    // 2. Private constructor to prevent instantiation
    private ConfigManager() {
        // Initialize configuration values
        System.out.println("ConfigManager instance created");
```

```
    }

    // 3. Public static method to get the instance (double-checked locking for
thread safety)
    public static ConfigManager getInstance() {
        if (instance == null) {
            synchronized (ConfigManager.class) {
                if (instance == null) {
                    instance = new ConfigManager();
                }
            }
        }
        return instance;
    }

    // Configuration storage
    private java.util.Map<String, String> config = new java.util.HashMap<>();

    // Business methods
    public String get(String key) {
        return config.get(key);
    }

    public void set(String key, String value) {
        config.put(key, value);
    }
}
```

EXPERIMENT 12

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.util.Random;

public class MultiThreadDemo {

    // Shared resource to demonstrate thread-safe operations
    private static final Object lock = new Object();
    private static int sharedCounter = 0;

    public static void main(String[] args) {
        // Create and start threads for different tasks
        Thread fileThread = new Thread(new FileReaderTask("sample.txt"),
"FileReader-Thread");
        Thread mathThread1 = new Thread(new MathComputationTask(), "MathThread-1");
        Thread mathThread2 = new Thread(new MathComputationTask(), "MathThread-2");

        fileThread.start();
        mathThread1.start();
        mathThread2.start();

        // Wait for all threads to complete
        try {
            fileThread.join();
            mathThread1.join();
```

```
            mathThread2.join();
        } catch (InterruptedException e) {
            System.out.println("Main thread interrupted");
        }

        System.out.println("\nAll threads completed. Final shared counter: " +
sharedCounter);
    }

    // Task for file reading
    static class FileReaderTask implements Runnable {
        private final String fileName;

        public FileReaderTask(String fileName) {
            this.fileName = fileName;
        }

        @Override
        public void run() {
            System.out.println(Thread.currentThread().getName() + " started reading
file: " + fileName);
            try (BufferedReader reader = new BufferedReader(new
FileReader(fileName))) {
                String line;
                while ((line = reader.readLine()) != null) {
                    System.out.println(Thread.currentThread().getName() + " read: "
+ line);
                    Thread.sleep(100); // Simulate processing time

                    // Thread-safe counter increment
                    synchronized (lock) {
                        sharedCounter++;
                    }
                }
            } catch (Exception e) {
                System.out.println(Thread.currentThread().getName() + " encountered
error: " + e.getMessage());
            }
            System.out.println(Thread.currentThread().getName() + " finished
reading file");
        }
    }

    // Task for mathematical computations
    static class MathComputationTask implements Runnable {
        private final Random random = new Random();

        @Override
        public void run() {
            System.out.println(Thread.currentThread().getName() + " started math
computations");
            for (int i = 0; i < 5; i++) {
                double num1 = random.nextDouble() * 100;
                double num2 = random.nextDouble() * 50;

                // Perform various computations
                double sum = num1 + num2;
                double product = num1 * num2;
                double sqrt = Math.sqrt(num1);
```

```
                System.out.printf("%s computed: %.2f + %.2f = %.2f, %.2f * %.2f =
%.2f, √%.2f = %.2f\n",
                    Thread.currentThread().getName(), num1, num2, sum, num1, num2,
product, num1, sqrt);

                try {
                    Thread.sleep(150); // Simulate processing time
                } catch (InterruptedException e) {
                    System.out.println(Thread.currentThread().getName() + " was
interrupted");
                }

                // Thread-safe counter increment
                synchronized (lock) {
                    sharedCounter++;
                }
            }
            System.out.println(Thread.currentThread().getName() + " finished
computations");
        }
    }
}

/*
To run this program, create a sample.txt file in the same directory with some text
content.
Example content for sample.txt:
Line 1: Hello from the file
Line 2: Multi-threading is powerful
Line 3: Java makes it easy
*/
```