### 1. *Factorial of a Number*

java

```
import java.util.Scanner;

public class Factorial {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a number: ");
        int n = sc.nextInt();
        long factorial = 1;

        for (int i = 1; i <= n; i++) {
            factorial *= i;
        }
        System.out.println("Factorial of " + n + " is: " + factorial);
    }
}
```

---

### 2. *First 50 Prime Numbers*

java

```
public class First50Primes {
    public static void main(String[] args) {
        int count = 0;
        int num = 2;

        while (count < 50) {
            if (isPrime(num)) {
                System.out.print(num + " ");
```

```
            count++;
        }
        num++;
    }
  }

  public static boolean isPrime(int n) {
    if (n <= 1) return false;
    for (int i = 2; i <= Math.sqrt(n); i++) {
      if (n % i == 0) return false;
    }
    return true;
  }
}
```

---

### 3. *Sum and Average of N Numbers*

java

```
import java.util.Scanner;

public class SumAndAverage {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.print("Enter the number of elements: ");
    int n = sc.nextInt();
    int[] numbers = new int[n];
    int sum = 0;

    System.out.println("Enter the numbers:");
```

```
        for (int i = 0; i < n; i++) {
            numbers[i] = sc.nextInt();
            sum += numbers[i];
        }
        double average = sum / (double) n;
        System.out.println("Sum: " + sum + ", Average: " + average);
    }
}
```

---

### 4. *Calculator with Simple Arithmetic Operations*

```
java
import java.util.Scanner;

public class Calculator {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter first number: ");
        double num1 = sc.nextDouble();
        System.out.print("Enter second number: ");
        double num2 = sc.nextDouble();
        System.out.print("Enter an operation (+, -, *, /): ");
        char op = sc.next().charAt(0);

        switch (op) {
            case '+':
                System.out.println("Result: " + (num1 + num2));
                break;
            case '-':
```

```
            System.out.println("Result: " + (num1 - num2));
            break;
        case '*':
            System.out.println("Result: " + (num1 * num2));
            break;
        case '/':
            if (num2 != 0) {
                System.out.println("Result: " + (num1 / num2));
            } else {
                System.out.println("Cannot divide by zero");
            }
            break;
        default:
            System.out.println("Invalid operation");
      }
   }
}
```

---

### 5. *Rectangle Class with Comparison*

```
java
class Rectangle {
   double width, length, area;
   String colour;

   public Rectangle(double width, double length, String colour) {
      this.width = width;
      this.length = length;
      this.colour = colour;
```

```
        this.area = width * length;
    }

    public double getLength() { return length; }
    public double getWidth() { return width; }
    public String getColour() { return colour; }
    public double findArea() { return area; }

    public boolean matches(Rectangle other) {
        return this.area == other.area && this.colour.equals(other.colour);
    }

    public static void main(String[] args) {
        Rectangle r1 = new Rectangle(4, 5, "Blue");
        Rectangle r2 = new Rectangle(4, 5, "Blue");

        if (r1.matches(r2)) {
            System.out.println("Matching Rectangles");
        } else {
            System.out.println("Non-matching Rectangles");
        }
    }
}
```

---

### 6. *Sort List of Integers*

```
java
import java.util.ArrayList;
import java.util.Collections;
```

```
import java.util.Scanner;

public class SortIntegers {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        ArrayList<Integer> list = new ArrayList<>();

        System.out.println("Enter integers (type 'done' to finish): ");
        while (sc.hasNextInt()) {
            list.add(sc.nextInt());
        }

        Collections.sort(list);
        System.out.println("Sorted List: " + list);
    }
}
```

---

### 7. *Sort List of Names*

```
java
import java.util.ArrayList;
import java.util.Collections;
import java.util.Scanner;

public class SortNames {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        ArrayList<String> names = new ArrayList<>();
```

```
        System.out.println("Enter names (type 'done' to finish): ");
        while (sc.hasNext()) {
            String name = sc.next();
            if (name.equalsIgnoreCase("done")) break;
            names.add(name);
        }

        Collections.sort(names);
        System.out.println("Sorted Names: " + names);
    }
}
```

---

### 8. *Add Two Matrices*

```
java
import java.util.Scanner;

public class AddMatrices {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter rows and columns of the matrix: ");
        int rows = sc.nextInt();
        int cols = sc.nextInt();

        int[][] matrix1 = new int[rows][cols];
        int[][] matrix2 = new int[rows][cols];
        int[][] sum = new int[rows][cols];

        System.out.println("Enter first matrix:");
```

```
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                matrix1[i][j] = sc.nextInt();
            }
        }

        System.out.println("Enter second matrix:");
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                matrix2[i][j] = sc.nextInt();
            }
        }

        System.out.println("Sum of matrices:");
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                sum[i][j] = matrix1[i][j] + matrix2[i][j];
                System.out.print(sum[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

--- Sure! I'll continue with the remaining problem statements.

---

### 9. *Method Overloading Demonstration*

java

```
public class MethodOverloading {
    public int add(int a, int b) {
        return a + b;
    }

    public double add(double a, double b) {
        return a + b;
    }

    public int add(int a, int b, int c) {
        return a + b + c;
    }

    public static void main(String[] args) {
        MethodOverloading mo = new MethodOverloading();
        System.out.println("add(int, int): " + mo.add(5, 10));
        System.out.println("add(double, double): " + mo.add(5.5, 10.2));
        System.out.println("add(int, int, int): " + mo.add(5, 10, 15));
    }
}
```

---

### 10. *Constructor Overloading Demonstration*

java

```
public class ConstructorOverloading {
    int x;
    double y;

    public ConstructorOverloading() {
```

```
        x = 0;
        y = 0.0;
    }

    public ConstructorOverloading(int x) {
        this.x = x;
        y = 0.0;
    }

    public ConstructorOverloading(int x, double y) {
        this.x = x;
        this.y = y;
    }

    public void display() {
        System.out.println("x: " + x + ", y: " + y);
    }

    public static void main(String[] args) {
        ConstructorOverloading c1 = new ConstructorOverloading();
        ConstructorOverloading c2 = new ConstructorOverloading(10);
        ConstructorOverloading c3 = new ConstructorOverloading(10, 20.5);

        c1.display();
        c2.display();
        c3.display();
    }
}
```

---

### 11. *Player Class with Inheritance*

java

```
class Player {
    String name;
    int age;

    public Player(String name, int age) {
        this.name = name;
        this.age = age;
    }
}

class CricketPlayer extends Player {
    public CricketPlayer(String name, int age) {
        super(name, age);
    }
}

class FootballPlayer extends Player {
    public FootballPlayer(String name, int age) {
        super(name, age);
    }
}

class HockeyPlayer extends Player {
    public HockeyPlayer(String name, int age) {
        super(name, age);
    }

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

```
        CricketPlayer cricketer = new CricketPlayer("Virat", 32);
        FootballPlayer footballer = new FootballPlayer("Lionel", 35);
        HockeyPlayer hockeyPlayer = new HockeyPlayer("Alex", 28);

        System.out.println("Cricket Player: " + cricketer.name);
        System.out.println("Football Player: " + footballer.name);
        System.out.println("Hockey Player: " + hockeyPlayer.name);
    }
}
```

---

### 12. *Exception Handling with Try and Catch*

```
java
import java.util.Scanner;

public class ExceptionHandling {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a number to divide: ");
        int num = sc.nextInt();

        try {
            System.out.print("Enter divisor: ");
            int divisor = sc.nextInt();
            int result = num / divisor;
            System.out.println("Result: " + result);
        } catch (ArithmeticException e) {
            System.out.println("Cannot divide by zero. " + e.getMessage());
        }
```

```
    }
}
```

---

### 13. *Drawing Shapes using Graphics Class*

```
java
import javax.swing.*;
import java.awt.*;

public class DrawShapes extends JPanel {
    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        g.drawRect(50, 50, 100, 50);  // Draw rectangle
        g.drawOval(200, 50, 100, 50); // Draw oval
        g.drawLine(50, 150, 200, 150); // Draw line
        g.drawString("Hello, Graphics!", 50, 200); // Draw text
    }

    public static void main(String[] args) {
        JFrame frame = new JFrame();
        DrawShapes panel = new DrawShapes();
        frame.add(panel);
        frame.setSize(400, 300);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

---

### 14. *Multiple Threads Communication*

java

```
class SharedResource {
    private int data;
    private boolean available = false;

    public synchronized void produce(int value) throws InterruptedException {
        while (available) wait();
        data = value;
        available = true;
        System.out.println("Produced: " + value);
        notify();
    }

    public synchronized int consume() throws InterruptedException {
        while (!available) wait();
        available = false;
        System.out.println("Consumed: " + data);
        notify();
        return data;
    }
}

class Producer extends Thread {
    private SharedResource shared;

    public Producer(SharedResource shared) {
        this.shared = shared;
    }
```

```
    public void run() {
        for (int i = 0; i < 5; i++) {
            try {
                shared.produce(i);
                Thread.sleep(100);
            } catch (InterruptedException e) { }
        }
    }
}

class Consumer extends Thread {
    private SharedResource shared;

    public Consumer(SharedResource shared) {
        this.shared = shared;
    }

    public void run() {
        for (int i = 0; i < 5; i++) {
            try {
                shared.consume();
                Thread.sleep(150);
            } catch (InterruptedException e) { }
        }
    }
}

public class ThreadCommunication {
    public static void main(String[] args) {
        SharedResource shared = new SharedResource();
```

```
        new Producer(shared).start();
        new Consumer(shared).start();
    }
}
```