

## Program 1

**Title:** Write a program in java to pass argument and count how many parameters are passed.

**Problem Description:** The program counts the number of parameters passed at the command prompt and displays them along with their occurrence.

**Method:** The method utilized here is to access the command line arguments using the args array in the main method. The length of this array is then used to determine the number of parameters passed.

### Code

```
public class Prg1 {  
    public static void main(String[] args) {  
        for (int i = 0; i < args.length; i++)  
        {  
            int count = 1;  
            for (int j = i + 1; j < args.length; j++)  
            {  
                if (args[i].equals(args[j]))  
                {  
                    count++;  
                    args[j] = ""; // Mark the counted argument to avoid duplicate counting  
                }  
            }  
            // Display the argument and its count if it's not already counted  
            if (!args[i].equals(""))  
            {  
                System.out.println(args[i] + ": " + count);  
            }  
        }  
    }  
}
```

## Program 2

**Title:** Write a java program to create and display the student details.

**Problem Description:** The problem involves creating a Java class named Student, which contains variables for USN, Name, Branch, and Phone. The task further requires writing a Java program to instantiate 'n' Student objects and printing their details with appropriate headings.

**Method:** In this program, the 'Student' class encapsulates the details of a student, and the main program creates an array of Student objects, initializes each object with sample data, and then print out the details of each student.

### Code

```
import java.util.Scanner;

class Student1 {
    String usn;
    String name;
    String branch;
    String phone;
    void read()
    {
        Scanner s=new Scanner(System.in);
        System.out.println("Enter usn");
        usn=s.next();
        System.out.println("Enter name");
        name=s.next();
        System.out.println("Enter branch");
        branch=s.next();
        System.out.println("Enter phone");
        phone=s.next();
    }
    // Method to display student details
    void display() {
        System.out.println("USN: " + usn);
```

```

        System.out.println("Name: " + name);
        System.out.println("Branch: " + branch);
        System.out.println("Phone: " + phone);
        System.out.println(); // Empty line for better formatting
    }
}

public class Student {
    public static void main(String[] args) {
        Scanner s=new Scanner(System.in);
        System.out.println("Enter number of students");
        int n=s.nextInt();
        Student1[] s1=new Student1[n];
        for (int i = 0; i < n; i++)
        {
            s1[i] = new Student1();
        }
        for(int i=0;i<n;i++)
        {
            s1[i].read();
        }
        for(int i=0;i<n;i++)
        {
            s1[i].display();
        }
    }
}

```

### **Program 3:**

**Title:** Write a java program to implement staff hierarchy.

**Problem Description:** The problem involves designing a superclass named Staff with attributes such as: StaffId, Name, Phone, and Salary. Additionally, three subclasses: Teaching,

Technical, and Contract-are to be created with specific attributes for each category. The task requires writing a Java program to read and display at least three staff objects from all three categories.

**Method:** In this example, the Staff superclass encapsulates common attributes shared by all staff members, while the Teaching, Technical, and Contract subclasses define specific attributes for each category of staff. The main program creates and displays at least three staff objects from each category.

### Code

```
import java.util.Scanner;
class Staff {
    String staffID, name, phone, salary;
    Scanner input = new Scanner(System.in);
    void read()
    {
        System.out.println("Enter StaffID");
        staffID = input.next();
        name = input.next();
        phone = input.next();
        salary = input.next();
    }
    void display()
    {
        System.out.print(staffID+"\t"+name+"\t"+phone+"\t"+salary+"\t");
    }
}
class Teaching extends Staff {
    String domain, publication;
    void read()
    {
        super.read(); // call super class read method
        System.out.println("Enter Domain");
        domain = input.next();
        System.out.println("Enter Publication");
        publication = input.next();
    }
    void display() {
        super.display(); // call super class display() method
        System.out.println(domain+"\t"+publication);
    }
}
class Technical extends Staff {
    String Skills;
    void read()
```

```

        {
            super.read(); // call super class read method
            System.out.println("Enter Skills");
            Skills = input.next();
        }
        void display()
        {
            super.display(); // call super class display() method
            System.out.println(Skills);
        }
    }
}
class Contract extends Staff
{
    String period;
    void read()
    {
        super.read(); // call super class read method
        System.out.println("Enter Period");
        period = input.next();
    }
    @Override
    void display() {
        super.display(); // call super class display() method
        System.out.println(period);
    }
}
}
public class StaffDemo
{
    public static void main(String[] args)
    {
        Scanner input = new Scanner(System.in);
        System.out.println("Enter number of staff details to be created");
        int n = input.nextInt();
        Teaching ts[] = new Teaching[n];
        Technical tech[] = new Technical[n];
        Contract c[] = new Contract[n];
        // Read Staff information under 3 categories
        for (int i = 0; i < n; i++)
        {
            System.out.println("Enter Teaching staff information");
            ts[i] = new Teaching();
            ts[i].read();
        }
        for (int i = 0; i < n; i++) {
            System.out.println("Enter Technical staff information");
            tech[i] = new Technical();
        }
    }
}

```

```

        tech[i].read();
    }
    for (int i = 0; i < n; i++) {
        System.out.println("Enter Contract staff information");
        c[i] = new Contract();
        c[i].read();
    }
    // Display Staff Information
    System.out.println("\n STAFF DETAILS: \n");
    System.out.println("-----TEACHING STAFF DETAILS----- ");
    for (int i = 0; i < n; i++)
        ts[i].display();
    System.out.println("-----TECHNICAL STAFF DETAILS-----");
    for (int i = 0; i < n; i++)
        tech[i].display();
    System.out.println("-----CONTRACT STAFF DETAILS-----");
    for (int i = 0; i < n; i++)
        c[i].display();

}
}

```

#### Program 4:

**Title:** Write a java program to implement Box class application to depict constructor overloading.

**Problem Description:** Given three classes called A, B, and C, C can be a subclass of B, which is a subclass of A. When this type of situation occurs, each subclass inherits all of the traits found in all of its super classes. methods with different parameters.

**Method:** In this program, C inherits all aspects of B and A. To see how a multilevel hierarchy can be useful, consider the following program. In it, the subclass BoxWeight is used as a superclass to create the subclass called Shipment. Shipment inherits all of the traits of BoxWeight and Box, and adds a field called cost, which holds the cost of shipping such a parcel.

**Theory Reference:** Module 3

#### Code

```

class Box {
    double length;
    double width;
    double height;

```

```

// Default constructor
Box() {
    length = -1;
    width = -1;
    height = -1;
}

// Parameterized constructor
Box(double len, double wid, double hgt) {
    length = len;
    width = wid;
    height = hgt;
}

// Copy constructor
Box(Box ob) {
    length = ob.length;
    width = ob.width;
    height = ob.height;
}

Box(double len)
{
    width=height=length=len;
}

// Method to calculate volume
double volume() {
    return length * width * height;
}
}

```

// Define the BoxWeight class that extends Box

**class** BoxWeight **extends** Box {

**double** weight;

    // Default constructor

    BoxWeight() {

**super**();

        weight = -1;

    }

    // Parameterized constructor

    BoxWeight(**double** len, **double** wid, **double** hgt, **double** wt) {

**super**(len, wid, hgt);

        weight = wt;

    }

    // Copy constructor

    BoxWeight(BoxWeight ob) {

**super**(ob);

        weight = ob.weight;

    }

    BoxWeight(**double** len, **double** wt)

    {

**super**(len);

        weight=wt;

    }

}

// Define the Shipment class that extends BoxWeight

**class** Shipment **extends** BoxWeight {

**double** cost;

    // Default constructor



```

Shipment() {
    super();
    cost = -1;
}

// Parameterized constructor
Shipment(double len, double wid, double hgt, double wt, double c) {
    super(len, wid, hgt, wt);
    cost = c;
}

// Copy constructor
Shipment(Shipment ob) {
    super(ob);
    cost = ob.cost;
}

Shipment(double len, double wt, double c)
{
    super(len,wt);
    cost=c;
}
}

public class BoxClassApplication
{
    public static void main(String[] args)
    {
        Shipment shipment1 =new Shipment(10, 20, 15, 10, 3.41);
        Shipment shipment2 =new Shipment(2, 3, 4, 0.76, 1.28);
        double vol;
        vol = shipment1.volume();
        System.out.println("Volume of shipment1 is " + vol);
        System.out.println("Weight of shipment1 is "+ shipment1.weight);
    }
}

```

```
System.out.println("Shipping cost: $" + shipment1.cost);  
System.out.println();  
vol = shipment2.volume();  
System.out.println("Volume of shipment2 is " + vol);  
System.out.println("Weight of shipment2 is "+ shipment2.weight);  
System.out.println("Shipping cost: $" + shipment2.cost);  
}  
}
```

**Program 5:**

**Title:** Write a java program to solve the Tower of Hanoi Problem using Stack.

**Problem Description:** Move all the disks stacked on the first tower over to the last tower using a helper tower in the middle. While moving the disks, certain rules must be followed. Only one disk can be moved. A larger disk cannot be placed on a smaller disk.

**Method:** In this program, Solving the Tower of Hanoi problem using a stack is an elegant and efficient approach. The Tower of Hanoi problem involves three rods and a number of disks of different sizes that can slide onto any rod. The puzzle starts with the disks in a neat stack in ascending order of size on one rod, with the smallest disk at the top, and the objective is to move the entire stack to another.

**Theory Reference:** Module 3 Page no:157

**Code:**

```
import java.util.*;

/* Class TowerOfHanoiUsingStacks */
public class p5
{
    public static int N;
    /* Creating Stack array */
    public static Stack<Integer>[] tower = new Stack[4];

    public static void main(String[] args)
    {
        Scanner scan = new Scanner(System.in);
        tower[1] = new Stack<Integer>();
        tower[2] = new Stack<Integer>();
        tower[3] = new Stack<Integer>();
        /* Accepting number of disks */
        System.out.println("Enter number of disks");
        int num = scan.nextInt();
        N = num;
        toh(num);
    }
}
```

```

/* Function to push disks into stack */
public static void toh(int n)
{
    for (int d = n; d > 0; d--)
        tower[1].push(d);
    display();
    move(n, 1, 2, 3);
}

/* Recursive Function to move disks */
public static void move(int n, int a, int b, int c)
{
    if (n > 0)
    {
        move(n-1, a, c, b);
        int d = tower[a].pop();
        tower[c].push(d);
        display();
        move(n-1, b, a, c);
    }
}

/* Function to display */
public static void display()
{
    System.out.println(" A | B | C");
    System.out.println("-----");
    for(int i = N - 1; i >= 0; i--)
    {
        String d1 = " ", d2 = " ", d3 = " ";
        try
        {
            d1 = String.valueOf(tower[1].get(i));
        }
        catch (Exception e){

```

```
    }  
    try  
    {  
        d2 = String.valueOf(tower[2].get(i));  
    }  
    catch(Exception e){  
    }  
    try  
    {  
        d3 = String.valueOf(tower[3].get(i));  
    }  
    catch (Exception e){  
    }  
    System.out.println(" "+d1+" | "+d2+" | "+d3);  
}  
System.out.println("\n");  
}  
}
```

## OUTPUT:

---

```
run:
Enter number of disks
2

  A | B | C
-----
  1 |   | 
  2 |   | 

  A | B | C
-----
    |   | 
  2 | 1 | 

  A | B | C
-----
    |   | 
    | 1 | 2

  A | B | C
-----
    |   | 1
    |   | 2

BUILD SUCCESSFUL (total time: 5 seconds)
```

**Program 6:**

**Title:** “Write a Java Program to calculate area and perimeter of variety of shapes (circle and triangle)”

**Problem Description:** Develop a JAVA program to create an abstract class Shape with abstract methods calculateArea() and calculatePerimeter(). Create subclasses Circle and Triangle that extend the Shape class and implement the respective methods to calculate the area and perimeter of each shape.

**Method:** Ensure that the program is well-structured, follows object-oriented principles, and provides clear and concise output demonstrating the functionality of each class and method.

**Theory Reference:** Module 3 Page no:157

**Code:**

```
import java.util.Scanner;

// Abstract class Shape
abstract class Shape {
    // Abstract methods
    abstract double calculateArea();
    abstract double calculatePerimeter();
}

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

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

    // Implementation of abstract methods
    @Override
    double calculateArea() {
        return Math.PI * radius * radius;
    }
}
```

```

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

// Subclass Triangle
class Triangle extends Shape {
    private double side1, side2, side3;

    // Constructor
    public Triangle(double side1, double side2, double side3) {
        this.side1 = side1;
        this.side2 = side2;
        this.side3 = side3;
    }

    // Implementation of abstract methods
    @Override
    double calculateArea() {
        // Using Heron's formula
        double s = (side1 + side2 + side3) / 2;
        return Math.sqrt(s * (s - side1) * (s - side2) * (s - side3));
    }

    @Override
    double calculatePerimeter() {
        return side1 + side2 + side3;
    }
}

// Main class to test the program
public class p6 {
    public static void main(String[] args) {
        // Circle with radius 5
        Scanner sc=new Scanner(System.in);

```



```

System.out.println("Enter the radius of circle");
int r=sc.nextInt();
Shape circle = new Circle(r);
System.out.println("Circle Area: " + circle.calculateArea());
System.out.println("Circle Perimeter: " + circle.calculatePerimeter());
System.out.println("Enter the sides of the triangle");
int s1=sc.nextInt();
int s2=sc.nextInt();
int s3=sc.nextInt();
Shape triangle = new Triangle(s1, s2, s3);
System.out.println("Triangle Area: " + triangle.calculateArea());
System.out.println("Triangle Perimeter: " + triangle.calculatePerimeter());
}
}

```

OUTPUT:

```

run:
Enter the radius of circle

5
Circle Area: 78.53981633974483
Circle Perimeter: 31.41592653589793
Enter the sides of the triangle
6
5
4
Triangle Area: 9.921567416492215
Triangle Perimeter: 15.0
BUILD SUCCESSFUL (total time: 16 seconds)

```

## Program 7:

**Title:** "Java Program: Resizable Interface for Object Resizing with Rectangle Implementation".

**Problem Description:** Develop a JAVA program to create an interface Resizable with methods `resizeWidth(int width)` and `resizeHeight(int height)` that allow an object to be resized. Create a class Rectangle that implements the Resizable interface and implements the resize methods.

**Method:** Ensure that the program follows proper object-oriented principles, such as encapsulation and abstraction, and provides clear and concise output demonstrating the resizing functionality.

**Theory Reference:** Module 3 Page no:157

### Code:

```
import java.util.Scanner;
// Interface Resizable
interface Resizable {
    void resizeWidth(int width);
    void resizeHeight(int height);
}
// Class Rectangle implementing Resizable
class Rectangle implements Resizable {
    private int width;
    private int height;
    // Constructor
    public Rectangle(int width, int height) {
        this.width = width;
        this.height = height;
    }
    // Method to resize the width
    @Override
    public void resizeWidth(int newWidth) {
        this.width = newWidth;
        System.out.println("New width: " + width);
    }
    // Method to resize the height
    @Override
    public void resizeHeight(int newHeight) {
        this.height = newHeight;
        System.out.println("New height: " + height);
    }
}
```

```
// Method to display the current dimensions of the rectangle
public void displayDimensions() {
    System.out.println("Current dimensions: " + width + " x " +
height);
}
}
// Main class to test the program
public class p7{
    public static void main(String[] args) {
        // Create a rectangle with width 10 and height 5
        Scanner sc=new Scanner(System.in);

        Rectangle rectangle = new Rectangle(10,5) ;
        rectangle.displayDimensions();
        // Resize the rectangle's width and height
        System.out.println("Enter the new width and height");
        int w=sc.nextInt();
        int h=sc.nextInt();
        rectangle.resizeWidth(w);
        rectangle.resizeHeight(h);
        rectangle.displayDimensions();
    }
}
```

## Program 8:

**Title:** "Java Program: Custom Exception Handling for DivisionByZero and Arithmetic Exceptions".

**Problem Description:** Develop a Java program that demonstrates the handling of custom exceptions, specifically for DivisionByZero and Arithmetic exceptions. You are required to utilize try-catch blocks along with throw statements to handle these exceptions gracefully.

**Method:** Ensure that the program demonstrates the proper usage of try-catch blocks, throw statements, and custom exception handling for DivisionByZero and Arithmetic exceptions.

**Theory Reference:** Module 4 Page no:205

### Code:

```
import java.util.Scanner;
//Custom exception class
class DivisionByZeroException extends Exception {
    public DivisionByZeroException(String message) {
        super(message);
    }
}

public class pgm8 {

    // Method to perform division and throw custom exception if
    denominator is zero
    static double divide(int numerator, int denominator) throws
    DivisionByZeroException {
        if (denominator == 0) {
            throw new DivisionByZeroException("Cannot divide by zero!");
        }
        return (double) numerator / denominator;
    }
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Scanner input = new Scanner(System.in);
        System.out.println("Enter numerator and denominator ");
        int numerator = input.nextInt();
        int denominator = input.nextInt();
        try {
            double result = divide(numerator, denominator);
            System.out.println("Result of division: " + result);
        }
    }
}
```

```
catch (DivisionByZeroException e) {  
    System.out.println("Exception caught: " + e.getMessage());  
}  
finally {  
    System.out.println("Finally block executed");  
}  
}
```

## Program 9

**Title:** "Write a Java program to generate random numbers using multiple threads.

**Problem Description:** Develop a Java program that implements a multi-threaded application with three threads. Each thread has a specific task as described as follows: First Thread (Random Number Generator): This thread generates a random integer every 1 second. Second Thread (Square Computation): This thread receives the random integer generated by the first thread and computes its square. After computing the square, it prints the result. Third Thread (Cube Computation): This thread receives the random integer generated by the first thread and computes its cube. After computing the cube, it prints the result.

**Method:** Program should demonstrate the multi-threading capabilities of Java and showcase the asynchronous computation of squares and cubes. Additionally, it should handle synchronization and data sharing effectively to prevent race conditions and ensure thread safety.

**Theory Reference:** Module 4 Page no:205

**Code:**

```
import java.util.Random;

class Square extends Thread
{
    int x;

    Square(int n)
    {
        x = n;
    }

    public void run()
    {
        int sqr = x * x;

        System.out.println("Square of " + x + " = " + sqr);
    }
}

class Cube extends Thread
{
    int x;

    Cube(int n)
    {
        x = n;
```

```

    }

    public void run()
    {
        int cub = x * x * x;

        System.out.println("Cube of " + x + " = " + cub);
    }
}

class Rnumber extends Thread
{
    public void run()
    {
        Random random = new Random();

        for (int i = 0; i < 5; i++) {
            int randomInteger = random.nextInt(10);

            System.out.println("Random Integer generated : " + randomInteger);

            Square s = new Square(randomInteger);
            s.start();

            Cube c = new Cube(randomInteger);
            c.start();

            try
            {
                Thread.sleep(1000);
            } catch (InterruptedException ex)
            {
                System.out.println(ex);
            }
        }
    }
}

public class P9_New {
    public static void main(String[] args)

```

```
{  
    Rnumber n = new Rnumber();  
    n.start();  
}  
}
```