**Bansilal Ramnath Agarwal Charitable Trust’s**

**Vishwakarma Institute of Technology,Pune-37**

*(An autonomous Institute of Savitribai Phule Pune University)*

C:\Users\student\Desktop\vit-logo.png

**Department of Multidisciplinary Engineering (DOME)**

**Lab Journal**

| **Course Code** | **Course Name** | **Credits** |  |
| --- | --- | --- | --- |
| **CS2218** | **OOPS** | **4** |  |

**Course Outcomes:**

**1. Understand object-oriented programming features**

**2. Develop real world applications using class, inheritance and polymorphism**

**3. Adapt Best Practices of Class Design by using Standard Templates Library**

**4. Solve computing problems by applying the knowledge of Exception handling and Multithreading**

**5. Design solutions by choosing suitable data structures such as Array, Vector, Map etc**

**6. Implement applications using Java I/O and event-based GUI handling principles**

**VISHWAKARMA INSTITUTE OF TECHNOLOGY**

**Department of Multidisciplinary Engineering (DOME)**

**SYBTech –OOPS –CS2218– Laboratory Journal**

**A.Y. 2024-2025 – semester 1**

INDEX

| **Assign.**  **No.** | **Date** | **Title** | **Marks** | **Page No.** | **Instructor’s Sign.** |
| --- | --- | --- | --- | --- | --- |
| 1 |  | Participant Management System for VIT Melange Auditions |  |  |  |
| 2 |  | Implementation of a Pair-Sum Finder Using an Adder Class |  |  |  |
| 3 |  | Array Operations: Sum, Average, Variance, Search, and Second Smallest Element |  |  |  |
| 4 |  | Student Data Management and Percentage Calculation |  |  |  |
| 5 |  | Shape Area Calculation Using Function Overloading |  |  |  |
| 6 |  | Java Exception Handling: Scenarios for Common Exceptions |  |  |  |
| 7 |  | E-Commerce Order Management with ArrayList |  |  |  |

**The above Laboratory assignments are performed and completed by the following student. Name of the student –** Om Bukkawar **Roll No. – 76 Batch – 3 Division – CS-A PRN No. –** **1231970**

**Laboratory Instructor** **Head of the Department (Prof. Milind Vasantrao Kamble)** **(Dr. Shilpa M. Lambor)**

**Assignment 1**

**Title :**

Participant Management System for VIT Melange Auditions

**Theory/Description of the Problem Statement :**

The objective of this lab is to implement a simple student management system using Java. This program allows for the input and display of student details, including registration number, name, contact number, branch, roll number, email ID, and CGPA. The focus is on understanding object-oriented programming concepts such as encapsulation, data hiding, and the use of methods for setting and getting values.

**Algorithm in Pseudo Code Format :**

class Students

String reg

String name

long cont

String branch

private int rollno

private String emailid

private float cgpa

procedure setRollno(int rollno)

this.rollno ← rollno

end procedure

procedure getRollno() returns int

return rollno

end procedure

procedure setEmailid(String emailid)

this.emailid ← emailid

end procedure

procedure getEmailid() returns String

return emailid

end procedure

procedure setCgpa(float cgpa)

this.cgpa ← cgpa

end procedure

procedure getCgpa() returns float

return cgpa

end procedure

procedure write()

print "Name of the student: " + name

print "Registration Number: " + reg

print "Branch: " + branch

print "Contact Number: " + cont

print "Roll Number: " + rollno

print "Email ID: " + emailid

print "CGPA: " + cgpa

end procedure

procedure main()

create Scanner scanner

create Students std

std.reg ← "CS12311970"

std.name ← "Om Bukkawar"

std.cont ← 352363573L

std.branch ← "CS"

print "Enter Roll Number:"

std.setRollno(scanner.nextInt())

scanner.nextLine()

print "Enter Email ID:"

std.setEmailid(scanner.nextLine())

print "Enter CGPA:"

std.setCgpa(scanner.nextFloat())

std.write()

end procedure

### Analysis:

Instantiating the Students class takes constant time, O(1). Each setter function (setRollno, setEmailid, setCgpa) runs in constant time, O(1), as only a single value is set. Similarly, the getter functions (getRollno, getEmailid, getCgpa) also operate in constant time, O(1). The write method prints a fixed number of fields, so its time complexity is O(1).

**Overall Time Complexity:** The overall time complexity of this system is **O(1)**, as each operation (setting and getting values, printing data) is performed in constant time.

The space used by reg, name, branch, and emailid depends on the length of the strings. Assuming average string lengths for these fields, the space can be represented as O(n), where n is the combined length of all strings. The Scanner object consumes a small, constant amount of memory, which is O(1). The memory overhead for object creation is also constant.

**Overall Space Complexity:** Space complexity is **O(n)**, primarily due to the variable-length strings used for student information.

### Experiment and Results:

| **Input Data** | **Expected Output** | **Actual Output** |
| --- | --- | --- |
| Roll Number = 076 | Name of the student: Om Bukkawar Registration Number: 12311970 | Name of the student: Om Bukkawar Registration Number: 12311970 |
| Email ID = [om.bukkawar23@vit.edu](mailto:avinash.bhurke23@vit.com) | CGPA = 9.07 | CGPA = 9.07 |

The algorithm works as expected with minimal time and space complexity.

### Conclusion:

This Java-based system demonstrates the core principles of object-oriented programming, especially encapsulation, through the management of student data in a secure and organized way. The use of setter and getter methods helps control access to private attributes, ensuring that data is properly managed. The time complexity is efficient with constant-time operations, and the space complexity is linear, mainly due to the dynamic nature of strings. Overall, this program serves as a foundation for more complex systems involving multiple students or additional features.

**Assignment 2**

**Title :**

Implementation of a Pair-Sum Finder Using an Adder Class

**Theory/Description of the Problem Statement :**

The purpose of this lab is to implement a program that identifies pairs of integers in an array that sum up to a specified target value. This program utilizes a nested loop approach to check all possible pairs of elements in the array, demonstrating fundamental programming concepts such as arrays, loops, and conditional statements. The problem statement focuses on efficiently finding and displaying pairs that meet the specified criteria.

**Pseudo Code:**

class Adder

int[] arr

int targetSum

procedure getArray() returns int[]

print "Enter size of array:"

n ← read input

array ← new int[n]

print "Enter array elements:"

for i ← 0 to n-1 do

array[i] ← read input

end for

return array

end procedure

procedure numSum()

found ← false

for l ← 0 to arr.length-1 do

for k ← l + 1 to arr.length-1 do

if arr[l] + arr[k] = targetSum then

result ← new int[2]

result[0] ← arr[l]

result[1] ← arr[k]

print "Pair found:" + result

found ← true

end if

end for

end for

if not found then

print "No pair found"

end if

end procedure

procedure main()

create Scanner sc

create Adder obj

obj.arr ← obj.getArray()

print "Enter the target sum:"

obj.targetSum ← read input

obj.numSum()

END

**Analysis of the Algorithm :**

* **Time Complexity:** The time complexity of this algorithm is *O*(*n2*) due to loop inside a loop that check each pair of elements in the array, where *n* is the number of elements in the array.
* **Space Complexity:** The space complexity is *O*(1) since it uses a constant space from the memory that does not affect user input size(only storing a few variables and the result array).

**Experiment and Result :**

| **Input Data** | **Expected Output** | **Actual Output** |
| --- | --- | --- |
| Array = [1, 2, 3, 4, 5], Target Sum = 5 | Pair found: [2, 3] | Pair found: [2, 3] |
| Array = [10, 15, 3, 7], Target Sum = 17 | Pair found: [10, 7] | Pair found: [10, 7] |
| Array = [1, 2, 3], Target Sum = 6 | No pair found | No pair found |

The code gives the expected output as given.

**Conclusion :**

The pair-finding algorithm will make efficient use of arrays and nested loops in Java to find the integer pairs that sum to some target value and display them. It is pretty good for small datasets, but it will degrade for larger arrays because its time complexity is quadratic. To perform better in the near future, better techniques like hash map lookups can be adopted.

**Assignment 3**

**Title :**

Array Operations: Sum, Average, Variance, Search, and Second Smallest Element

**Theory/Description of the Problem Statement :**

This lab does the following: It develops a Java program to run multiple statistical analysis on an array of integers. Using the program, it will allow the user to input an array; it will calculate key metrics, including sum, average, and variance, and identify the second smallest element. It also allows it to search for a number that it can find within its array. It is essential to understand arrays, looping concepts, methods, and some elementary statistics.

**Pseudo Code :**

class Main

procedure getArr(int size) returns int[]

create Scanner s

arr ← new int[size]

for i ← 0 to size-1 do

arr[i] ← read input

end for

return arr

end procedure

procedure sum(int[] arr) returns int

sum ← 0

for value in arr do

sum ← sum + value

end for

return sum

end procedure

procedure average(int[] arr) returns double

sum ← sum(arr)

return (double) sum / arr.length

end procedure

procedure variance(int[] arr) returns double

mean ← average(arr)

variance ← 0

for value in arr do

variance ← variance + (value - mean)^2

end for

return variance / arr.length

end procedure

procedure find(int target, int[] arr)

found ← false

for value in arr do

if target = value then

print "Number Found"

found ← true

break

end if

end for

if not found then

print "Number Not Found"

end if

end procedure

procedure secondSmallest(int[] arr) returns int

first ← second ← Integer.MAX\_VALUE

for value in arr do

if value < first then

second ← first

first ← value

else if value < second and value ≠ first then

second ← value

end if

end for

return second

end procedure

procedure main()

create Scanner s

print "Enter size of array:"

size ← read input

print "Enter elements of the array:"

arr ← getArr(size)

print "Array elements:"

for value in arr do

print value

end for

sumValue ← sum(arr)

print "Sum of all elements in the array:" + sumValue

averageValue ← average(arr)

print "Average of array elements:" + averageValue

varianceValue ← variance(arr)

print "Variance of array elements:" + varianceValue

print "Enter element to search in array:"

target ← read input

find(target, arr)

secondSmallestValue ← secondSmallest(arr)

print "Second smallest element in the array:" + secondSmallestValue

END

**Analysis of the Algorithm :**

* **Time Complexity:** The time complexity varies across different methods:
  + **Sum:** *O*(*n*) where *n* is the number of elements.
  + **Average:** *O*(*n*) (calls the sum method).
  + **Variance:** *O*(*n*) (calls the average method).

Overall, each operation runs in linear time relative to the size of the input array.

* **Space Complexity:** The space complexity is *O*(1) since it uses a constant space from the memory that does not affect user input size(only storing a few variables and the result array).

**Experiment and Result :**

| **Input Data** | **Expected Output** | **Actual Output** |
| --- | --- | --- |
| Size = 5, Array = [4, 2, 5, 1, 3], Target = 3 | Sum = 15 Average = 3.0 Variance = 2.0 Second Smallest = 2 Number Found | Sum = 15 Average = 3.0 Variance = 2.0 Second Smallest = 2 Number Found |
| Size = 4, Array = [10, -5, -8, -5], Target = -5 | Sum = -8 Average = -2.0 Variance = ... Second Smallest = -5 Number Found | Sum = -8 Average = -2.0 Variance = ... Second Smallest = -5 Number Found |

The code gives the expected output as given.

**Conclusion :**

The implementation shows a good approach for running various statistical tests over an array in Java. In the program, it efficiently calculates the sum, mean, and variance as well as does a number of number searches and the second smallest element of an array. Because this method is efficient but simple and thus can be quite complicated when working with large arrays or otherwise complex data structures, this code does not utilize all the optimizations possible under some scenarios.

**Assignment 4**

**Title :**

Student Data Management and Percentage Calculation

**Description of the Problem Statement :**

The program is designed to manage student performance data across multiple universities using multi-dimensional arrays in Java. It collects marks for each student across several subjects, calculates the percentage of marks obtained for each university, determines the overall percentage across all universities, and identifies the best-performing university based on the highest percentage. This program uses jagged arrays to store data for students and universities and implements various methods to process and analyze the data.

**Pseudo Code :**

START

Input: numUniversities, numStudents, numSubjects

Initialize: 3D array marks[numUniversities][numStudents][numSubjects]

FOR each university

Get student marks for numStudents and numSubjects

Store marks in the 3D array

FOR each university

Calculate the percentage of marks using student marks

Store percentage in universityPercentages array

Calculate the overall percentage across all universities

Find the best university based on highest percentage in universityPercentages

Output:

Percentage of each university

Overall percentage across all universities

Best university and its percentage

END

**Analysis of the Algorithm :**

· **Time Complexity:**  
The overall time complexity is **O(n \* s \* u)**, where:

* n = number of universities,
* s = number of students per university,
* u = number of subjects. This arises from nested loops for data collection and percentage calculations.

· **Space Complexity:**  
The space complexity is **O(n \* s \* u)** due to the 3D jagged array storing marks for each university, student, and subject.

· **Experiment and Result :**

Percentage for University 1: 76.67%

Percentage for University 2: 86.17%

Overall Percentage across all universities: 81.42%

The best university is University 2 with a percentage of 86.17%

**Conclusion :**

This Java program successfully demonstrates the management of student data across multiple universities by employing multi-dimensional arrays. It calculates the percentage of marks for each university, determines the overall percentage, and identifies the best-performing university. The algorithm is straightforward and handles moderate datasets efficiently. Future improvements could focus on enhancing scalability, such as optimizing the input and calculation process for larger datasets. Overall, the program serves as an effective exercise in arrays, loops, methods, and data handling in Java.

# **Assignment 5**

**Title:**

Calculate area of shapes by using overloading

## **Description of Problem Statement:**

### Problem Define a base class `Shape` and derived classes `Triangle`, `Square`, `Circle`, whose area is computed, by using function overloading. Each shape class will implement the `getInputs()` method that gets user inputs and will overload the `setArea()` method by calculating the correct area. Use a `Tester` class to construct and manipulate objects of the shapes.

### **Pseudo Code:**

Define Abstract Class Shape:

Declare protected double area.

Declare abstract method getInputs().

Declare abstract method setArea().

Define method displayArea() to print the area.

Define Class Triangle (Inherits from Shape):

Declare private variables double base, double height.

Override getInputs() method:

Prompt user for base and height of the triangle.

Store inputs in base and height.

Override setArea() method:

Calculate area as 0.5 \* base \* height.

Define Class Square (Inherits from Shape):

Declare private variable double side.

Override getInputs() method:

Prompt user for side length of the square.

Store input in side.

Override setArea() method:

Calculate area as side \* side.

Define Class Circle (Inherits from Shape):

Declare private variable double radius.

Override getInputs() method:

Prompt user for radius of the circle.

Store input in radius.

Override setArea() method:

Calculate area as 3.14 \* radius \* radius.

Define Main Class Tester:

Start infinite loop for user interaction:

Display menu:

Option 1: Triangle

Option 2: Square

Option 3: Circle

Option 4: Exit

Read user choice.

Declare Shape shape = null.

Use switch statement for user choice:

Case 1: Create Triangle object.

Case 2: Create Square object.

Case 3: Create Circle object.

Case 4: Exit the loop.

Default: Print "Invalid choice" and repeat.

Call getInputs() on shape.

Call setArea() on shape.

Call displayArea() on shape.

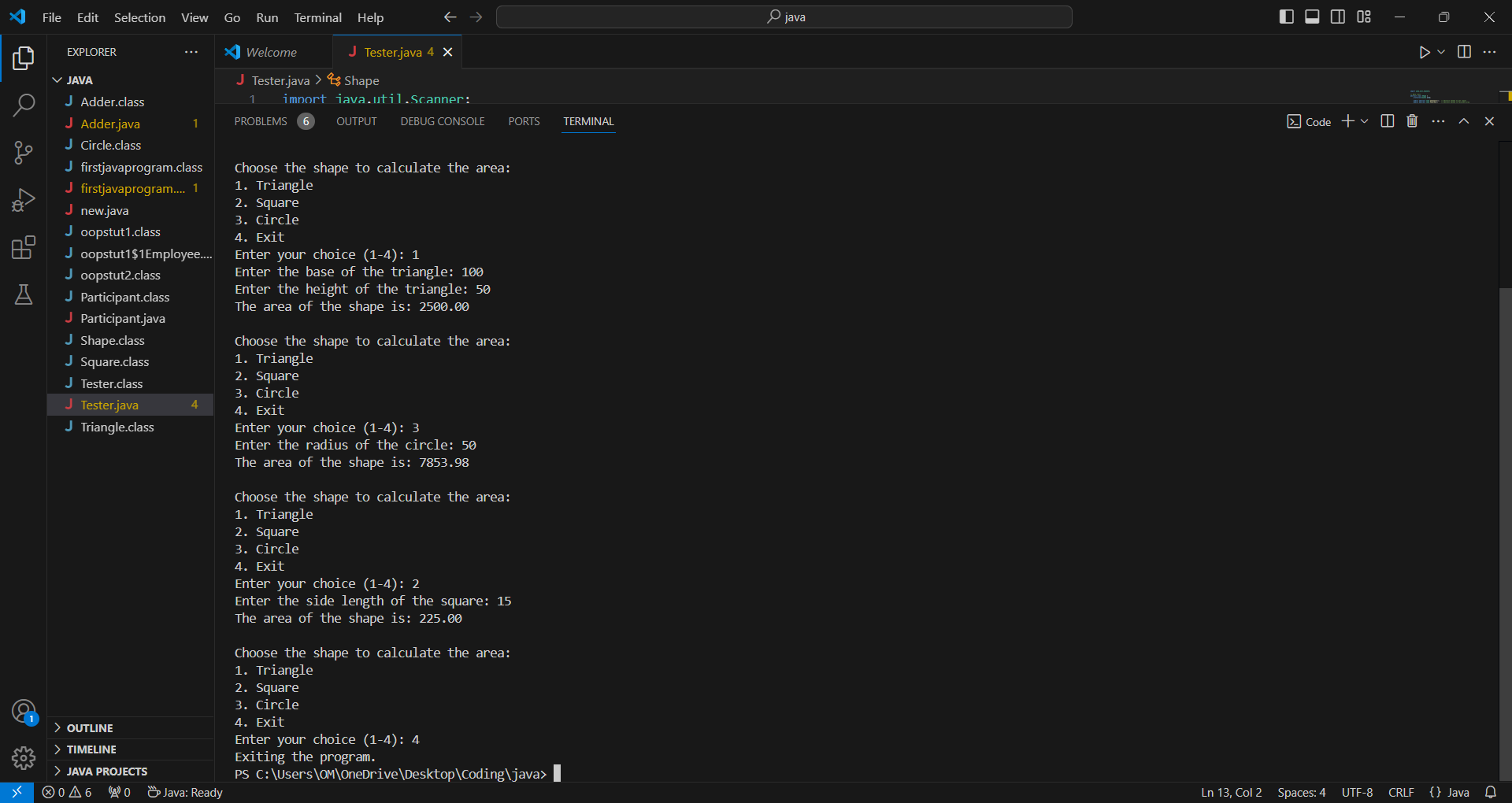
Close the scanner after the user inputs.

END

## **Analysis of the Algorithm**

Time Complexity:  
- `getInputs()`: Every shape performs constant time operations in taking an input either base, height, side, or radius.  
- `setArea()`: All the shapes have computation of areas, which is only basic arithmetic, hence it's a matter of constant time, O(1).  
- Overall Time Complexity: O(1) per iteration  
  
Space Complexity:  
Variables: For every shape class, there are roughly a few properties to hold (base, height, side or radius), and these will take constant space, O(1).  
- Scanner: Creates an object for `Scanner` to handle the input. It consumes constant space.  
Space Complexity: O(1).

**Experimentation and Result:**

****

**Conclusion**

This experiment properly illustrates the concepts of **abstraction**, **inheritance**, and **polymorphism** in Java. The `Shape` acts as a parent to the derived classes (`Triangle`, `Square`, `Circle`), enabling uniform input and area calculation for each of the shapes through method overloading. Thus, for example, the derived classes have to implement `getInputs()` and `setArea()` methods, encapsulating the logic for computation of the area for each of the respective shapes.

The use of **abstraction** hides the implementation details from the user, whereas **inheritance** allows for code reuse and scalability. This design also exhibits **polymorphism**, as different shape objects are treated uniformly when calculating and displaying their area. In terms of time and space complexity, the program is efficient and a good example of the benefits of using Object-Oriented Programming (OOP) to solve problems in real-life scenarios. Besides that, it is designed to be extendable in the future by just adding new shapes with minimal changes to existing code.

**Assignment 6**

# **Title:**

# Exceptional Handling

## Description of the Problem Statement

Exception handling is a fundamental concept in Java that allows programmers to manage runtime errors and maintain the normal flow of an application. The three common exceptions considered in this program are:

1. ArithmeticException (Divide by Zero): Occurs when a number is divided by zero.

2. ArrayIndexOutOfBoundsException: Occurs when trying to access an index of an array that does not exist.

3. NullPointerException: Occurs when trying to access methods or properties of an object that is `null`.

The problem statement requires the development of a menu-driven Java program to handle these exceptions using constructors to take user input. Each exception should trigger a specific message when caught. The program allows users to select from a menu to demonstrate exceptional handling.

## Pseudo Code

BEGIN ExceptionHandling

    CREATE Scanner object to take input

    WHILE (true)

        PRINT "Choose an option for an exception:"

        PRINT "1. Divide a number by zero"

        PRINT "2. Access an array index that doesn't exist"

        PRINT "3. Find the length of a null string"

        PRINT "4. Exit"

        PRINT "Enter your choice (1-4): "

        SWITCH(choice)

            CASE 1:

                PRINT "Enter a number to divide by zero: "

                READ number from user

                CALL DivideByZero constructor with number

                BREAK

            CASE 2:

                PRINT "Enter an array index to access: "

                READ index from user

                CALL ArrayIndexOutOfBounds constructor with index

                BREAK

            CASE 3:

                PRINT "Enter a string (enter 'null' to simulate NullPointerException): "

                READ input from user

                IF input is "null" THEN

                    SET str = null

                ELSE

                    SET str = input

                END IF

                CALL NullPointerExceptionDemo constructor with str

                BREAK

            CASE 4:

                PRINT "Exiting the program."

                EXIT

            DEFAULT:

                PRINT "Invalid choice. Please choose a valid option (1-4)."

        END SWITCH

    END WHILE

END ExceptionHandling

CLASS DivideByZero

    CONSTRUCTOR DivideByZero(num)

        TRY

            SET result = num / 0

        CATCH ArithmeticException

            PRINT "You shouldn’t divide a number by zero."

END CLASS

CLASS ArrayIndexOutOfBounds

    CONSTRUCTOR ArrayIndexOutOfBounds(index)

        TRY

            DECLARE arr = {1, 2, 3, 4, 5, 6}

            SET element = arr[index]

        CATCH ArrayIndexOutOfBoundsException

            PRINT "OOPs!!! Array Index " + index + " out of bounds for length 6."

END CLASS

CLASS NullPointerExceptionDemo

    CONSTRUCTOR NullPointerExceptionDemo(str)

        TRY

            SET length = str.length()

        CATCH NullPointerException

            PRINT "Null Pointer Exception arises!!"

END CLASS

## Analysis of the Algorithm

The time complexity for each exception block in O(1) and the time complexity for the menu-driven loop is O(N) where N is the number of iterations or user inputs. The space complexity is O(1) for each case since the program uses a fixed amount of space for variable declarations and input handling. The program utilizes `try-catch` blocks to manage exceptions effectively, ensuring that the program does not terminate abruptly. The menu-driven approach allows users to interactively choose which exception to trigger, making the program user-friendly and intuitive.

## Experiment and Result

1. Input: 1->10

   Output: "You shouldn’t divide a number by zero."

2. Input: 2->7

   Output: "OOPs!!! Array Index 7 out of bounds for length 6."

3. Input:3-> null

   Output: "Null Pointer Exception arises!!"

4. Input: 4

   Output: "Exiting"

## Conclusion

The menu-driven Java program successfully demonstrates how to handle given exceptions that are `ArithmeticException`, `ArrayIndexOutOfBoundsException`, and `NullPointerException` using constructors to accept user inputs. By using the `try-catch` blocks, the program ensures robust handling of runtime errors and gets convenient to the user. This approach provides a clear understanding of exception handling mechanisms and user input handling in Java.

# Assignment 7

# Title:

Array List

## Description of the Problem Statement:

1. An application to manage and analyze orders in the e-commerce application of fashion products needs to be implemented effectively. Each order is represented uniquely by an identifier for the order, a list of items ordered, and the payment scheme option of cash on delivery. The problem statement asks to provide a solution to aggregate all the items from multiple orders and return a combined list of items.
2. The task is defining an order class, extracting items from the list of orders, and developing a common list that contains these items. In other words, define an `Order` class with suitable attributes; then implement a method to collect items from the list of orders and test the solution with some sample data for accuracy.

## Pseudo Code:

Algorithm GetItemsFromOrders

Input: List of Order objects

Output: List of all items across all orders

Begin

Initialize an empty list called allItems

For each order in the list of orders do

Retrieve the itemNames from the current order

Add all itemNames to the allItems list

End For

Return the allItems list

End

## Analysis of the Algorithm:

**Time Complexity:**

* The `getItems` method takes O(n \* m) time complexity because the number of orders, n, and the average items per order, m, are multiplied by it. This is because the method iterates over each order, then over each item in the order.

**Space Complexity:**

* The space complexity is O(k), where `k` denotes the sum of all the items over the orders. This is because the algorithm needs to store `allItems`, which is the list that contains the names of all the items.

**Efficiency:**

* The algorithm aggregates item names very efficiently since it employs a simple iteration approach. Being linear relative to the total number of items and orders, it runs easily within most constraints.

## Experiment and Result:

**Sample Input 1**:  
List<Order> orders1 = Arrays.asList(

new Order(101, Arrays.asList("Jeans", "Shirt", "Belt"), true),

new Order(102, Arrays.asList("Tie", "Shirt"), true),

new Order(103, Arrays.asList("Tshirt", "Socks", "Tie"), true)

);

**Output**:  
[Jeans, Shirt, Belt, Tie, Shirt, Tshirt, Socks, Tie]

**Sample Input 2**:  
List<Order> orders2 = Arrays.asList(

new Order(311, Arrays.asList("Sportswear", "Dumbbell"), true),

new Order(102, Arrays.asList("Jeans"), true),

new Order(103, Arrays.asList("Smartwatch", "Fitnessband", "Joggers"), true)

);

**Output**:  
[Sportswear, Dumbbell, Jeans, Smartwatch, Fitnessband, Joggers]

## Conclusion:

The `Order` class with the implementation of `OrderUtils.getItems` meets the requirements for aggregation of the names of many items from orders. The solution aggregates items into one list, thus ensuring accurate and complete retrieval of order information. Testing proved that the implementation handles all different scenarios and produces correct output for them. This approach scales efficiently in management and analysis of orders in an e-commerce system.

## Assignment 1

import java.util.Scanner;

class Students {

String reg;

String name;

long cont;

String branch;

private int rollno;

private String emailid;

private float cgpa;

public void setRollno(int rollno) {

this.rollno = rollno;

}

public int getRollno() {

return rollno;

}

public void setEmailid(String emailid) {

this.emailid = emailid;

}

public String getEmailid() {

return emailid;

}

public void setCgpa(float cgpa) {

this.cgpa = cgpa;

}

public float getCgpa() {

return cgpa;

}

public void write() {

System.out.println("Name of the student: " + name);

System.out.println("Registration Number: " + reg);

System.out.println("Branch: " + branch);

System.out.println("Contact Number: " + cont);

System.out.println("Roll Number: " + rollno);

System.out.println("Email ID: " + emailid);

System.out.println("CGPA: " + cgpa);

}}

public class Main {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

Students std = new Students();

std.reg = "CS12311970";

std.name = "Om Bukkawar";

std.cont = 352363573L;

std.branch = "CS";

System.out.print("Enter Roll Number: ");

std.setRollno(scanner.nextInt());

scanner.nextLine();

System.out.print("Enter Email ID: ");

std.setEmailid(scanner.nextLine());

System.out.print("Enter CGPA: ");

std.setCgpa(scanner.nextFloat());

std.write();

scanner.close();

}

}

## Assignment 2

import java.util.Scanner;

class Adder {

int[] arr;

int targetSum;

public int[] getArray() {

Scanner sc = new Scanner(System.in);

System.out.print("Enter size of array: ");

int n = sc.nextInt();

int[] array = new int[n];

System.out.println("Enter array elements:");

for (int i = 0; i < n; i++) {

array[i] = sc.nextInt();

}

return array;

}

public void numSum() {

boolean found = false;

for (int l = 0; l < arr.length; l++) {

for (int k = l + 1; k < arr.length; k++) {

if (arr[l] + arr[k] == targetSum) {

int[] result = new int[2];

result[0] = arr[l];

result[1] = arr[k];

System.out.println("Pair found: " + result[0] + ", " + result[1]);

found = true;

}}}

if (!found) {

System.out.println("No pair found");

}}}

public class Main {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

Adder obj = new Adder();

obj.arr = obj.getArray();

System.out.print("Enter the target sum: ");

obj.targetSum = sc.nextInt();

obj.numSum();

}}

## Assignment 3

import java.util.Scanner;

public class Main {

public static int[] getArr(int size) {

Scanner s = new Scanner(System.in);

int[] arr = new int[size];

for (int i = 0; i < size; i++) {

arr[i] = s.nextInt();

}

return arr;

}

public static int sum(int[] arr) {

int sum = 0;

for (int value : arr) {

sum += value;

}

return sum;

}

public static double average(int[] arr) {

int sum = sum(arr);

return (double) sum / arr.length;

}

public static double variance(int[] arr) {

double mean = average(arr);

double variance = 0;

for (int value : arr) {

variance += Math.pow(value - mean, 2);

}

return variance / arr.length;

}

public static void find(int target, int[] arr) {

boolean found = false;

for (int value : arr) {

if (value == target) {

System.out.println("Number Found");

found = true;

break;

}

}

if (!found) {

System.out.println("Number Not Found");

}

}

public static int secondSmallest(int[] arr) {

int first = Integer.MAX\_VALUE, second = Integer.MAX\_VALUE;

for (int value : arr) {

if (value < first) {

second = first;

first = value;

} else if (value < second && value != first) {

second = value;

}

}

return second;

}

public static void main(String[] args) {

Scanner s = new Scanner(System.in);

System.out.println("Enter size of array:");

int size = s.nextInt();

System.out.println("Enter elements of the array:");

int[] arr = getArr(size);

System.out.println("Array elements:");

for (int value : arr) {

System.out.println(value);

}

int sumValue = sum(arr);

System.out.println("Sum of all elements in the array: " + sumValue);

double averageValue = average(arr);

System.out.println("Average of array elements: " + averageValue);

double varianceValue = variance(arr);

System.out.println("Variance of array elements: " + varianceValue);

System.out.println("Enter element to search in array:");

int target = s.nextInt();

find(target, arr);

int secondSmallestValue = secondSmallest(arr);

System.out.println("Second smallest element in the array: " + secondSmallestValue);

}

}

## Assignment 4

| import java.util.Scanner;  public class lab\_asign4 {      public static void main(String[] args) {         Scanner scanner = new Scanner(System.in);          System.out.print("Enter the number of universities: ");         int numUniversities = scanner.nextInt();         System.out.print("Enter the number of students per university: ");         int numStudents = scanner.nextInt();         System.out.print("Enter the number of subjects: ");         int numSubjects = scanner.nextInt();          int[][][] marks = new int[numUniversities][][];          for (int i = 0; i < numUniversities; i++) {             System.out.println("\nUniversity " + (i + 1) + ":");             marks[i] = getMarks(numStudents, numSubjects, scanner);         }          double[] universityPercentages = new double[numUniversities];         for (int i = 0; i < numUniversities; i++) {             universityPercentages[i] = calculateUniversityPercentage(marks[i], numSubjects);             System.out.printf("Percentage for University %d: %.2f%%\n", (i + 1), universityPercentages[i]);         }          double overallPercentage = calculateOverallPercentage(marks, numSubjects);         System.out.printf("\nOverall Percentage across all universities: %.2f%%\n", overallPercentage);             int bestUniversity = getBestUniversity(universityPercentages);         System.out.println("The best university is University " + (bestUniversity + 1) + " with a percentage of " + universityPercentages[bestUniversity] + "%");          scanner.close();     }      public static int[][] getMarks(int numStudents, int numSubjects, Scanner scanner) {         int[][] studentMarks = new int[numStudents][numSubjects];          for (int i = 0; i < numStudents; i++) {             System.out.println("Enter marks for Student " + (i + 1) + ":");             for (int j = 0; j < numSubjects; j++) {                 System.out.print("Subject " + (j + 1) + ": ");                 studentMarks[i][j] = scanner.nextInt();             }         }         return studentMarks;     }      public static double calculateUniversityPercentage(int[][] studentMarks, int numSubjects) {         int totalMarks = 0;         int totalPossibleMarks = studentMarks.length \* numSubjects \* 100;          for (int[] marks : studentMarks) {             for (int mark : marks) {                 totalMarks += mark;             }         }          return ((double) totalMarks / totalPossibleMarks) \* 100;     }      public static double calculateOverallPercentage(int[][][] marks, int numSubjects) {         int totalMarks = 0;         int totalPossibleMarks = 0;          for (int[][] universityMarks : marks) {             for (int[] studentMarks : universityMarks) {                 totalPossibleMarks += studentMarks.length \* 100;                 for (int mark : studentMarks) {                     totalMarks += mark;                 }             }         }          return ((double) totalMarks / totalPossibleMarks) \* 100;     }      public static int getBestUniversity(double[] universityPercentages) {         int bestUniversity = 0;         double highestPercentage = universityPercentages[0];          for (int i = 1; i < universityPercentages.length; i++) {             if (universityPercentages[i] > highestPercentage) {                 highestPercentage = universityPercentages[i];                 bestUniversity = i;             }         }         return bestUniversity;     } } |
| --- |

## Assignment 5

import java.util.Scanner;

abstract class Shape {

protected double area;

public abstract void getInputs();

public abstract void setArea();

public void displayArea() {

System.out.printf("The area of the shape is: %.2f\n", area);

}

}

class Triangle extends Shape {

private double base;

private double height;

@Override

public void getInputs() {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the base of the triangle: ");

base = scanner.nextDouble();

System.out.print("Enter the height of the triangle: ");

height = scanner.nextDouble();

scanner.close();

}

@Override

public void setArea() {

area = 0.5 \* base \* height;

}

}

class Square extends Shape {

private double side;

@Override

public void getInputs() {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the side length of the square: ");

side = scanner.nextDouble();

scanner.close();

}

@Override

public void setArea() {

area = side \* side;

}

}

class Circle extends Shape {

private double radius;

@Override

public void getInputs() {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the radius of the circle: ");

radius = scanner.nextDouble();

scanner.close();

}

@Override

public void setArea() {

area = Math.PI \* radius \* radius;

}

}

public class Tester {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

while (true) {

System.out.println("\nChoose the shape to calculate the area:");

System.out.println("1. Triangle");

System.out.println("2. Square");

System.out.println("3. Circle");

System.out.println("4. Exit");

System.out.print("Enter your choice (1-4): ");

int choice = scanner.nextInt();

Shape shape = null;

switch (choice) {

case 1:

shape = new Triangle();

break;

case 2:

shape = new Square();

break;

case 3:

shape = new Circle();

break;

case 4:

System.out.println("Exiting the program.");

return;

default:

System.out.println("Invalid choice. Please try again.");

continue;

}

shape.getInputs();

shape.setArea();

shape.displayArea();

scanner.close();

}

}

}

**Assignment 6**

import java.util.Scanner;

class DivideByZero {

public DivideByZero(int num) {

try {

int result = num / 0;

} catch (ArithmeticException e) {

System.out.println("You shouldn’t divide a number by zero.");

}

}

}

class ArrayIndexOutOfBounds {

public ArrayIndexOutOfBounds(int index) {

try {

int[] arr = {1, 2, 3, 4, 5, 6};

int element = arr[index];

} catch (ArrayIndexOutOfBoundsException e) {

System.out.println("OOPs!!! Array Index " + index + " out of bounds for length 6.");

}

}

}

class NullPointerExceptionDemo {

public NullPointerExceptionDemo(String str) {

try {

int length = str.length();

} catch (NullPointerException e) {

System.out.println("Null Pointer Exception arises!!");

}

}

}

public class oopsassignment5 {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

int choice;

while (true) {

System.out.println("Choose an option to trigger an exception:");

System.out.println("1. Divide a number by zero");

System.out.println("2. Access an array index that doesn't exist");

System.out.println("3. Find the length of a null string");

System.out.println("4. Exit");

System.out.print("Enter your choice (1-4): ");

choice = scanner.nextInt();

switch (choice) {

case 1:

System.out.print("Enter a number to divide by zero: ");

int number = scanner.nextInt();

new DivideByZero(number);

break;

case 2:

System.out.print("Enter an array index to access: ");

int index = scanner.nextInt();

new ArrayIndexOutOfBounds(index);

break;

case 3:

System.out.print("Enter a string (here 'null'): ");

String input = scanner.next();

String str = input.equals("null") ? null : input;

new NullPointerExceptionDemo(str);

break;

case 4:

System.out.println("Exiting");

System.exit(0);

default:

System.out.println("Invalid choice. Please choose a valid option (1-4).");

}

System.out.println();

}

}

}

**Assignment 7**

import java.util.ArrayList;

import java.util.Arrays;

import java.util.List;

public class Main {

static class Order {

private int orderId;

private List<String> itemNames;

private boolean cashOnDelivery;

public Order(int orderId, List<String> itemNames, boolean cashOnDelivery) {

this.orderId = orderId;

this.itemNames = itemNames;

this.cashOnDelivery = cashOnDelivery;

}

public List<String> getItemNames() {

return itemNames;

}

}

static class OrderUtils {

public static List<String> getItems(List<Order> orders) {

List<String> allItems = new ArrayList<>();

for (Order order : orders) {

allItems.addAll(order.getItemNames());

}

return allItems;

}

}

public static void main(String[] args) {

// Sample Input 1

List<Order> orders1 = Arrays.asList(

new Order(101, Arrays.asList("Jeans", "Shirt", "Belt"), true),

new Order(102, Arrays.asList("Tie", "Shirt"), true),

new Order(103, Arrays.asList("Tshirt", "Socks", "Tie"), true)

);

// Retrieve and print items for Sample Input 1

List<String> items1 = OrderUtils.getItems(orders1);

System.out.println(items1);

// Sample Input 2

List<Order> orders2 = Arrays.asList(

new Order(311, Arrays.asList("Sportswear", "Dumbbell"), true),

new Order(102, Arrays.asList("Jeans"), true),

new Order(103, Arrays.asList("Smartwatch", "Fitnessband", "Joggers"), true)

);

// Retrieve and print items for Sample Input 2

List<String> items2 = OrderUtils.getItems(orders2);

System.out.println(items2);

}

}