1. Write the java program in Arithmetic exception for the following

a) Divided by Zero

b) Logarithm of negative or Zero

c) Tan 90 Degree

d) Zero power Zero

public class ArithmeticExceptionsDemo {

public static void main(String[] args) {

// Divided by Zero

try {

int result = 10 / 0;

} catch (ArithmeticException e) {

System.out.println("ArithmeticException: " + e.getMessage());

}

// Logarithm of negative or Zero

try {

double logResult = Math.log(-10);

} catch (ArithmeticException e) {

System.out.println("ArithmeticException: " + e.getMessage());

}

// Tan 90 Degree

try {

double tanResult = Math.tan(Math.toRadians(90));

} catch (ArithmeticException e) {

System.out.println("ArithmeticException: " + e.getMessage());

}

// Zero power Zero

try {

double powerResult = Math.pow(0, 0);

} catch (ArithmeticException e) {

System.out.println("ArithmeticException: " + e.getMessage());

}

}

}

2.Write the java program in Array Index Out Of Bound exception for the following

a) Accessing an array element outside its bound

b) Iterating beyond the array length

c) Nested array and incorrect index

d) Passing incorrect array to the method

public class ArrayExceptionsDemo {

public static void main(String[] args) {

int[] array = {1, 2, 3};

// Accessing an array element outside its bound

try {

int element = array[3];

} catch (ArrayIndexOutOfBoundsException e) {

System.out.println("ArrayIndexOutOfBoundsException: " + e.getMessage());

}

// Iterating beyond the array length

try {

for (int i = 0; i <= array.length; i++) {

System.out.println(array[i]);

}

} catch (ArrayIndexOutOfBoundsException e) {

System.out.println("ArrayIndexOutOfBoundsException: " + e.getMessage());

}

// Nested array and incorrect index

try {

int[][] nestedArray = {{1, 2}, {3, 4}};

int element = nestedArray[1][2];

} catch (ArrayIndexOutOfBoundsException e) {

System.out.println("ArrayIndexOutOfBoundsException: " + e.getMessage());

}

// Passing incorrect array to the method

try {

int[] incorrectArray = null;

printArrayLength(incorrectArray);

} catch (ArrayIndexOutOfBoundsException e) {

System.out.println("ArrayIndexOutOfBoundsException: " + e.getMessage());

}

}

public static void printArrayLength(int[] arr) {

System.out.println("Array length: " + arr.length);

}

}

3.Write the java program in Number format Exception for the following

a) Parsing a Non-Numeric String.

b) Reading User Input Without Validation.

c) Formatting Issues in “Decimal Format”.

d) Incorrectly using localized decimal separators

public class NumberFormatExceptionDemo {

public static void main(String[] args) {

// Parsing a Non-Numeric String

try {

String nonNumeric = "abc";

int number = Integer.parseInt(nonNumeric);

} catch (NumberFormatException e) {

System.out.println("NumberFormatException: " + e.getMessage());

}

// Reading User Input Without Validation

try {

String userInput = "123xyz";

int number = Integer.parseInt(userInput);

} catch (NumberFormatException e) {

System.out.println("NumberFormatException: " + e.getMessage());

}

// Formatting Issues in "Decimal Format"

try {

String formattedNumber = "12,345.67";

double number = Double.parseDouble(formattedNumber);

} catch (NumberFormatException e) {

System.out.println("NumberFormatException: " + e.getMessage());

}

// Incorrectly using localized decimal separators

try {

String localizedNumber = "12.345,67";

double number = Double.parseDouble(localizedNumber.replace(",", ""));

} catch (NumberFormatException e) {

System.out.println("NumberFormatException: " + e.getMessage());

}

}

}

4. Write the java program in Illegal Argument Exception for the following

a) Negative Argument in the method requiring nonnegative values

b) Invalid Enum Constant Passed to a Method

c) Setting an Invalid Range for a Method Parameter

d) Empty or Null String Argument in a Method Requiring

public class IllegalArgumentExceptionDemo {

public static void main(String[] args) {

// Negative Argument in the method requiring non-negative values

try {

int negativeValue = -10;

validatePositiveValue(negativeValue);

} catch (IllegalArgumentException e) {

System.out.println("IllegalArgumentException: " + e.getMessage());

}

// Invalid Enum Constant Passed to a Method

try {

String color = "Pink";

validateColor(color);

} catch (IllegalArgumentException e) {

System.out.println("IllegalArgumentException: " + e.getMessage());

}

// Setting an Invalid Range for a Method Parameter

try {

int age = 150;

validateAge(age);

} catch (IllegalArgumentException e) {

System.out.println("IllegalArgumentException: " + e.getMessage());

}

// Empty or Null String Argument in a Method Requiring Non-Empty String

try {

String name = null;

validateName(name);

} catch (IllegalArgumentException e) {

System.out.println("IllegalArgumentException: " + e.getMessage());

}

}

public static void validatePositiveValue(int value) {

if (value < 0) {

throw new IllegalArgumentException("Value must be non-negative");

}

}

public static void validateColor(String color) {

if (!color.equalsIgnoreCase("Red") && !color.equalsIgnoreCase("Blue") && !color.equalsIgnoreCase("Green")) {

throw new IllegalArgumentException("Invalid color");

}

}

public static void validateAge(int age) {

if (age < 0 || age > 120) {

throw new IllegalArgumentException("Invalid age range");

}

}

public static void validateName(String name) {

if (name == null || name.isEmpty()) {

throw new IllegalArgumentException("Name cannot be null or empty");

}

}

}

5. Throw a custom exception, if an user try to enter

negative values

public class NegativeValueException extends Exception {

public NegativeValueException(String message) {

super(message);

}

public static void main(String[] args) {

try {

int value = -10;

if (value < 0) {

throw new NegativeValueException("Negative values are not allowed");

}

} catch (NegativeValueException e) {

System.out.println("Error: " + e.getMessage());

}

}

}

6. Throw a custom exception, if an user try to enter invalid decimal number

public class InvalidDecimalException extends Exception {

public InvalidDecimalException(String message) {

super(message);

}

public static void main(String[] args) {

try {

String decimal = "12.34.56";

if (!decimal.matches("\\d+(\\.\\d+)?")) {

throw new InvalidDecimalException("Invalid decimal number format");

}

} catch (InvalidDecimalException e) {

System.out.println("Error: " + e.getMessage());

}

}

}

7. Throw a custom exception, if an user try to enter an non-integer value

public class NonIntegerException extends Exception {

public NonIntegerException(String message) {

super(message);

}

public static void main(String[] args) {

try {

String userInput = "abc";

if (!userInput.matches("\\d+")) {

throw new NonIntegerException("Non-integer values are not allowed");

}

} catch (NonIntegerException e) {

System.out.println("Error: " + e.getMessage());

}

}

}

1. Use throws keyword at class and method level

import java.io.IOException;

public class ThrowsExample {

public static void main(String[] args) throws IOException {

throwIOException();

}

public static void throwIOException() throws IOException {

throw new IOException("IOException occurred");

}

}

9. Extend Thread class and create two threads, T1 for Armstrong numbers and T2 for Fibonacci series, both threads should use all thread life cycle methods

public class ArmstrongThread extends Thread {

@Override

public void run() {

System.out.println("ArmstrongThread started");

// Logic for Armstrong numbers

for (int i = 1; i <= 1000; i++) {

if (isArmstrong(i)) {

System.out.println(i + " is an Armstrong number");

}

}

System.out.println("ArmstrongThread completed");

}

private boolean isArmstrong(int num) {

int originalNum, remainder, result = 0;

originalNum = num;

while (originalNum != 0) {

remainder = originalNum % 10;

result += Math.pow(remainder, 3);

originalNum /= 10;

}

return result == num;

}

public static void main(String[] args) {

ArmstrongThread t1 = new ArmstrongThread();

FibonacciThread t2 = new FibonacciThread();

t1.start();

t2.start();

}

}

class FibonacciThread extends Thread {

@Override

public void run() {

System.out.println("FibonacciThread started");

// Logic for Fibonacci series

int n = 10, t1 = 0, t2 = 1;

System.out.print("Fibonacci Series: ");

for (int i = 1; i <= n; ++i) {

System.out.print(t1 + " ");

int sum = t1 + t2;

t1 = t2;

t2 = sum;

}

System.out.println();

System.out.println("FibonacciThread completed");

}

}

10. create 2 threads in java, and find which thread has longer waiting and sleeping time

public class ThreadTimeComparison {

public static void main(String[] args) {

Thread t1 = new Thread(() -> {

System.out.println("Thread 1 started");

try {

Thread.sleep(5000); // Sleep for 5 seconds

} catch (InterruptedException e) {

e.printStackTrace();

}

System.out.println("Thread 1 completed");

});

Thread t2 = new Thread(() -> {

System.out.println("Thread 2 started");

try {

Thread.sleep(3000); // Sleep for 3 seconds

} catch (InterruptedException e) {

e.printStackTrace();

}

System.out.println("Thread 2 completed");

});

long startTime = System.currentTimeMillis();

t1.start();

t2.start();

try {

t1.join();

t2.join();

} catch (InterruptedException e) {

e.printStackTrace();

}

long endTime = System.currentTimeMillis();

System.out.println("Total time taken: " + (endTime - startTime) + " milliseconds");

}

}

11. Join 2 threads if they contain even numbers or put them in sleep state. That is T1 for 10000ms and T2 for 5000ms

public class JoinThreads {

public static void main(String[] args) {

Thread t1 = new Thread(() -> {

System.out.println("Thread 1 started");

for (int i = 1; i <= 10; i++) {

if (i % 2 == 0) {

System.out.println("Thread 1: " + i);

}

}

System.out.println("Thread 1 completed");

});

Thread t2 = new Thread(() -> {

System.out.println("Thread 2 started");

for (int i = 1; i <= 10; i++) {

if (i % 2 != 0) {

System.out.println("Thread 2: " + i);

}

}

System.out.println("Thread 2 completed");

});

t1.start();

t2.start();

try {

if (t1.isAlive() && t2.isAlive()) {

t1.join(10000); // Join with a timeout of 10000 milliseconds

t2.join(5000); // Join with a timeout of 5000 milliseconds

}

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

12. Create a Threads T1, T2 both with 3x3 matrix, Thread 3 should wait for Thread 4 to complete the count of elements in both T1 and T2, Then T3 can acquire T1 and T2 and complete matrix addition

public class MatrixOperations {

private static int[][] matrix1 = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};

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

private static int[][] resultMatrix = new int[3][3];

public static void main(String[] args) {

Thread t1 = new Thread(() -> {

System.out.println("Thread 1 started");

countElements(matrix1);

System.out.println("Thread 1 completed");

});

Thread t2 = new Thread(() -> {

System.out.println("Thread 2 started");

countElements(matrix2);

System.out.println("Thread 2 completed");

});

Thread t3 = new Thread(() -> {

try {

t1.join(); // Wait for Thread 1 to complete

t2.join(); // Wait for Thread 2 to complete

} catch (InterruptedException e) {

e.printStackTrace();

}

System.out.println("Thread 3 started");

addMatrices(matrix1, matrix2);

System.out.println("Thread 3 completed");

});

t1.start();

t2.start();

t3.start();

}

private static void countElements(int[][] matrix) {

int count = 0;

for (int[] row : matrix) {

for (int num : row) {

count++;

}

}

System.out.println("Number of elements in matrix: " + count);

}

private static void addMatrices(int[][] matrix1, int[][] matrix2) {

for (int i = 0; i < matrix1.length; i++) {

for (int j = 0; j < matrix1[i].length; j++) {

resultMatrix[i][j] = matrix1[i][j] + matrix2[i][j];

}

}

System.out.println("Matrix Addition Result:");

for (int[] row : resultMatrix) {

for (int num : row) {

System.out.print(num + " ");

}

System.out.println();

}

}

}