**a) Concept of Java Exception and Exception Handling**

In Java, an exception is an event that disrupts the normal flow of a program's execution. Exceptions are typically caused by errors or unexpected conditions that occur during runtime, such as dividing by zero, accessing an array out of bounds, or trying to read a non-existent file.

Exception handling in Java is a mechanism to handle these runtime errors gracefully so that the program can either:

1. Recover from the error
2. Provide a meaningful error message to the user
3. Clean up resources before terminating

Java provides keywords for exception handling:

* try: Block of code to monitor for exceptions
* catch: Block that handles specific exceptions
* finally: Block that always executes (for cleanup)
* throw: Used to explicitly throw an exception
* throws: Declares exceptions a method might throw

Java exceptions are organized in a hierarchy with Throwable at the top, with two main subclasses:

* Error (serious problems not meant to be caught)
* Exception (conditions that should be caught)

**b) Exception for Division by Zero.**

When a Java program attempts to divide air pollutant levels by zero during average AQI calculation, it throws an Arithmetic Exception with the message "/ by zero".

This exception is triggered because division by zero is mathematically undefined. In computing, attempting this operation would lead to an undefined or infinite result, which cannot be represented in computer memory. Java (and most programming languages) explicitly prevent this operation by throwing an exception to alert the programmer about the invalid operation.

Example;

int totalPollutants = 100;

int numberOfDays = 0;

double averageAQI = totalPollutants / numberOfDays; // Throws ArithmeticException

## c) while vs for loops in Java

Both while and for loops are used for iteration in Java, but they have different use cases:

1. **while loop**: Best when the number of iterations is unknown beforehand. It checks the condition before each iteration.

// Simulating NEMA tracking PM2.5 levels for 30 days with while loop

int day = 1;

while (day <= 30) {

double pm25 = getPM25Reading(); // hypothetical method

System.out.println("Day " + day + " PM2.5: " + pm25);

day++;

}

1. **for loop**: Best when the number of iterations is known. It combines initialization, condition, and increment in one line.

// Simulating NEMA tracking PM2.5 levels for 30 days with for loop

for (int day = 1; day <= 30; day++) {

double pm25 = getPM25Reading(); // hypothetical method

System.out.println("Day " + day + " PM2.5: " + pm25);

}

Key differences:

* for loops are more concise when iterating a known number of times
* while loops are better when the termination condition is more complex or not based on a counter
* for loops keep the loop control variables within the loop scope

## d) Java Program for AQI Analysis

import java.util.Arrays;

import java.util.Random;

public class AQIMonitor {

public static void main(String[] args) {

// i) Generate 30 random AQI readings between 1 and 300

int[] aqiReadings = generateAQIReadings(30);

System.out.println("Generated AQI Readings:");

System.out.println(Arrays.toString(aqiReadings));

// ii) Compute and display the median AQI value

double median = calculateMedian(aqiReadings);

System.out.printf("\nMedian AQI: %.1f\n", median);

// iii) Identify and count the number of "hazardous" days (AQI > 200)

int hazardousDays = countHazardousDays(aqiReadings);

System.out.println("Number of hazardous days (AQI > 200): " + hazardousDays);

}

// Generate random AQI readings between 1 and 300

public static int[] generateAQIReadings(int days) {

Random random = new Random();

int[] readings = new int[days];

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

readings[i] = random.nextInt(300) + 1; // 1-300 inclusive

}

return readings;

}

// Calculate median AQI value

public static double calculateMedian(int[] readings) {

int[] sorted = readings.clone();

Arrays.sort(sorted);

int length = sorted.length;

if (length % 2 == 0) {

return (sorted[length/2 - 1] + sorted[length/2]) / 2.0;

} else {

return sorted[length/2];

}

}

// Count days with AQI > 200 (hazardous)

public static int countHazardousDays(int[] readings) {

int count = 0;

for (int aqi : readings) {

if (aqi > 200) {

count++;

}

}

return count;

}

}