1. Arithmetic Exception

public class ArithmeticExceptionDemo {

public static void main(String[] args) {

// a) Divided by Zero

try {

int a = 10;

int result = a / 0; // This will cause ArithmeticException

} catch (ArithmeticException e) {

System.out.println("Error: Divided by zero");

}

// b) Logarithm of negative or zero

try {

double result = Math.log(-1); // This will return NaN, no exception but might require handling.

if (Double.isNaN(result)) {

throw new ArithmeticException("Error: Logarithm of negative or zero");

}

} catch (ArithmeticException e) {

System.out.println(e.getMessage());

}

// c) Tan 90 Degree (approximation as Math.tan takes radians)

try {

double result = Math.tan(Math.toRadians(90)); // Large value close to infinity.

if (Double.isInfinite(result)) {

throw new ArithmeticException("Error: Tan of 90 degrees leads to infinity");

}

} catch (ArithmeticException e) {

System.out.println(e.getMessage());

}

// d) Zero power Zero

try {

double result = Math.pow(0, 0); // Indeterminate form.

System.out.println("0^0 is undefined or indeterminate");

} catch (ArithmeticException e) {

System.out.println("Error: Zero power Zero");

}

}

}

2. Array Index Out of Bounds Exception

java

public class ArrayIndexOutOfBoundsExceptionDemo {

public static void main(String[] args) {

// a) Accessing an array element outside its bound

try {

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

int value = arr[3]; // Accessing out of bounds

} catch (ArrayIndexOutOfBoundsException e) {

System.out.println("Error: Array index out of bounds");

}

// b) Iterating beyond the array length

try {

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

for (int i = 0; i <= arr.length; i++) { // Will go out of bounds

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

}

} catch (ArrayIndexOutOfBoundsException e) {

System.out.println("Error: Iterated beyond array length");

}

// c) Nested array and incorrect index

try {

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

int value = nestedArr[1][2]; // Accessing an invalid index in nested array

} catch (ArrayIndexOutOfBoundsException e) {

System.out.println("Error: Incorrect index in nested array");

}

// d) Passing incorrect array to the method

try {

int[] arr = null;

processArray(arr); // Passing null array

} catch (ArrayIndexOutOfBoundsException e) {

System.out.println("Error: Passed incorrect array to the method");

}

}

public static void processArray(int[] arr) {

System.out.println(arr[0]); // May cause exception if array is null

}

}

3. Number Format Exception

java

import java.text.DecimalFormat;

import java.util.Scanner;

public class NumberFormatExceptionDemo {

public static void main(String[] args) {

// a) Parsing a Non-Numeric String

try {

int value = Integer.parseInt("abc"); // Will throw NumberFormatException

} catch (NumberFormatException e) {

System.out.println("Error: Parsing a non-numeric string");

}

// b) Reading User Input Without Validation

try {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter a number: ");

String input = scanner.nextLine();

int number = Integer.parseInt(input); // No validation, can cause exception

} catch (NumberFormatException e) {

System.out.println("Error: User input is not a valid number");

}

// c) Formatting Issues in “Decimal Format”

try {

DecimalFormat df = new DecimalFormat("#.##");

String formatted = df.format("abc"); // Will throw exception

} catch (IllegalArgumentException e) {

System.out.println("Error: Invalid format in DecimalFormat");

}

// d) Incorrectly using localized decimal separators

try {

String number = "1,23"; // Using comma instead of dot for decimal separator

double value = Double.parseDouble(number); // Will throw NumberFormatException

} catch (NumberFormatException e) {

System.out.println("Error: Incorrect use of localized decimal separators");

}

}

}

4. Illegal Argument Exception

java

public class IllegalArgumentExceptionDemo {

public static void main(String[] args) {

// a) Negative Argument in a method requiring nonnegative values

try {

sqrt(-5); // Will throw IllegalArgumentException

} catch (IllegalArgumentException e) {

System.out.println(e.getMessage());

}

// b) Invalid Enum Constant Passed to a Method

try {

printDay(null); // Will throw IllegalArgumentException

} catch (IllegalArgumentException e) {

System.out.println(e.getMessage());

}

// c) Setting an Invalid Range for a Method Parameter

try {

setRange(10, 5); // Invalid range

} catch (IllegalArgumentException e) {

System.out.println(e.getMessage());

}

// d) Empty or Null String Argument in a method requiring non-empty strings

try {

printMessage(""); // Will throw IllegalArgumentException

} catch (IllegalArgumentException e) {

System.out.println(e.getMessage());

}

}

public static double sqrt(double value) {

if (value < 0) {

throw new IllegalArgumentException("Error: Negative argument in sqrt method");

}

return Math.sqrt(value);

}

public static void printDay(String day) {

if (day == null) {

throw new IllegalArgumentException("Error: Null value for day");

}

System.out.println("Day is: " + day);

}

public static void setRange(int start, int end) {

if (start > end) {

throw new IllegalArgumentException("Error: Invalid range, start cannot be greater than end");

}

System.out.println("Valid range: " + start + " to " + end);

}

public static void printMessage(String message) {

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

throw new IllegalArgumentException("Error: Null or empty string");

}

System.out.println("Message: " + message);

}

}

Here are the requested Java programs with custom exceptions, thread usage, and specific conditions:

5. Custom Exception for Negative Values

class NegativeValueException extends Exception {

public NegativeValueException(String message) {

super(message);

}

}

public class CustomExceptionDemo1 {

public static void checkValue(int value) throws NegativeValueException {

if (value < 0) {

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

}

System.out.println("Valid value: " + value);

}

public static void main(String[] args) {

try {

checkValue(-5); // Will throw NegativeValueException

} catch (NegativeValueException e) {

System.out.println(e.getMessage());

}

}

}

```

6. Custom Exception for Invalid Decimal Numbers

class InvalidDecimalException extends Exception {

public InvalidDecimalException(String message) {

super(message);

}

}

public class CustomExceptionDemo2 {

public static void checkDecimal(String input) throws InvalidDecimalException {

try {

Double.parseDouble(input);

} catch (NumberFormatException e) {

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

}

System.out.println("Valid decimal: " + input);

}

public static void main(String[] args) {

try {

checkDecimal("12.a3"); // Will throw InvalidDecimalException

} catch (InvalidDecimalException e) {

System.out.println(e.getMessage());

}

}

}

```

7. Custom Exception for Non-Integer Values

class NonIntegerValueException extends Exception {

public NonIntegerValueException(String message) {

super(message);

}

}

public class CustomExceptionDemo3 {

public static void checkInteger(String input) throws NonIntegerValueException {

try {

Integer.parseInt(input);

} catch (NumberFormatException e) {

throw new NonIntegerValueException("Non-integer value entered");

}

System.out.println("Valid integer: " + input);

}

public static void main(String[] args) {

try {

checkInteger("abc"); // Will throw NonIntegerValueException

} catch (NonIntegerValueException e) {

System.out.println(e.getMessage());

}

}

}

```

8. Using `throws` at Class and Method Level

class CustomExceptionDemo4 {

public static void validate(int number) throws Exception {

if (number < 0) {

throw new Exception("Negative number exception");

}

System.out.println("Valid number: " + number);

}

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

validate(-10); // Throws exception at method level

}

}

```

9. Extending `Thread` Class for Armstrong Numbers and Fibonacci Series

class ArmstrongThread extends Thread {

public void run() {

System.out.println("Armstrong numbers between 100 and 500:");

for (int num = 100; num <= 500; num++) {

int sum = 0, temp = num, remainder;

while (temp != 0) {

remainder = temp % 10;

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

temp /= 10;

}

if (sum == num) {

System.out.println(num);

}

}

}

}

class FibonacciThread extends Thread {

public void run() {

System.out.println("Fibonacci series up to 100:");

int a = 0, b = 1, c;

System.out.print(a + " " + b);

while (b <= 100) {

c = a + b;

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

a = b;

b = c;

}

System.out.println();

}

}

public class ThreadDemo {

public static void main(String[] args) {

ArmstrongThread t1 = new ArmstrongThread();

FibonacciThread t2 = new FibonacciThread();

t1.start(); // Start the Armstrong number thread

t2.start(); // Start the Fibonacci series thread

}

}

```

10. Find Which Thread Has Longer Waiting and Sleeping Time

class WaitingThread extends Thread {

public void run() {

try {

System.out.println(Thread.currentThread().getName() + " is waiting...");

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

} catch (InterruptedException e) {

e.printStackTrace();

}

System.out.println(Thread.currentThread().getName() + " has finished waiting.");

}

}

class SleepingThread extends Thread {

public void run() {

try {

System.out.println(Thread.currentThread().getName() + " is sleeping...");

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

} catch (InterruptedException e) {

e.printStackTrace();

}

System.out.println(Thread.currentThread().getName() + " has finished sleeping.");

}

}

public class ThreadSleepWait {

public static void main(String[] args) {

WaitingThread t1 = new WaitingThread();

SleepingThread t2 = new SleepingThread();

t1.start();

t2.start();

}

}

```

11. Join Threads with Even Numbers or Sleep

class ThreadEvenNumbers extends Thread {

private int sleepTime;

public ThreadEvenNumbers(int sleepTime) {

this.sleepTime = sleepTime;

}

public void run() {

for (int i = 2; i <= 10; i += 2) {

System.out.println(i);

}

try {

Thread.sleep(sleepTime);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

public class JoinThreadsDemo {

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

ThreadEvenNumbers t1 = new ThreadEvenNumbers(10000); // Sleep for 10000ms

ThreadEvenNumbers t2 = new ThreadEvenNumbers(5000); // Sleep for 5000ms

t1.start();

t2.start();

t1.join(); // Join t1

t2.join(); // Join t2

}

}

```

12. Thread with Matrix Addition and Synchronization

class MatrixThread extends Thread {

int[][] matrix;

int count;

public MatrixThread(int[][] matrix) {

this.matrix = matrix;

this.count = 0;

}

public void run() {

countElements();

}

private void countElements() {

for (int[] row : matrix) {

for (int element : row) {

count++;

}

}

System.out.println("Thread " + Thread.currentThread().getName() + " counted elements: " + count);

}

}

public class MatrixAdditionDemo {

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

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

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

MatrixThread t1 = new MatrixThread(matrix1);

MatrixThread t2 = new MatrixThread(matrix2);

t1.start();

t2.start();

t1.join();

t2.join();

Thread t3 = new Thread(() -> {

System.out.println("Adding matrices...");

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 3; j++) {

System.out.print((matrix1[i][j] + matrix2[i][j]) + " ");

}

System.out.println();

}

});

t3.start();

t3.join();

}

}