1. public class ArithmeticExceptions {

public static void main(String[] args) {

// Divided by Zero

try {

int result = 10 / 0;

System.out.println("Result: " + result);

} catch (ArithmeticException e) {

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

}

// Logarithm of negative or Zero

try {

double result = Math.log(-1);

System.out.println("Result: " + result);

} catch (ArithmeticException e) {

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

}

// Tan 90 Degree

try {

double result = Math.tan(Math.PI / 2);

System.out.println("Result: " + result);

} catch (ArithmeticException e) {

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

}

// Zero power Zero

try {

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

System.out.println("Result: " + result);

} catch (ArithmeticException e) {

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

}

}

}

2. public class ArrayIndexExceptions {

public static void main(String[] args) {

// Accessing an array element outside its bound

try {

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

System.out.println(arr[3]);

} catch (ArrayIndexOutOfBoundsException e) {

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

}

// Iterating beyond the array length

try {

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

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

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

}

} catch (ArrayIndexOutOfBoundsException e) {

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

}

// Nested array and incorrect index

try {

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

System.out.println(nestedArray[1][3]);

} catch (ArrayIndexOutOfBoundsException e) {

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

}

// Passing incorrect array to the method

try {

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

printArray(arr, 4);

} catch (ArrayIndexOutOfBoundsException e) {

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

}

}

static void printArray(int[] arr, int index) {

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

}

}

3. import java.text.DecimalFormat;

import java.text.ParseException;

public class NumberFormatExceptions {

public static void main(String[] args) {

// Parsing a Non-Numeric String

try {

String nonNumericString = "abc";

int number = Integer.parseInt(nonNumericString);

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

} catch (NumberFormatException e) {

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

}

// Reading User Input Without Validation

try {

String userInput = "123abc";

int number = Integer.parseInt(userInput);

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

} catch (NumberFormatException e) {

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

}

// Formatting Issues in "Decimal Format"

try {

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

String formattedNumber = df.format(123.45);

int number = Integer.parseInt(formattedNumber);

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

} catch (NumberFormatException e) {

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

}

// Incorrectly using localized decimal separators

try {

String localizedNumber = "1,234,567.89"; // Using comma as a decimal separator

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

Number parsedNumber = df.parse(localizedNumber);

System.out.println("Number: " + parsedNumber);

} catch (ParseException e) {

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

}

}

}

4. public class IllegalArgumentExceptions {

public static void main(String[] args) {

// Negative Argument in the method requiring nonnegative values

try {

int radius = -5;

double area = calculateCircleArea(radius);

System.out.println("Area: " + area);

} catch (IllegalArgumentException e) {

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

}

// Invalid Enum Constant Passed to a Method

try {

printMonth(13); // Passing an invalid month number

} catch (IllegalArgumentException e) {

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

}

// Setting an Invalid Range for a Method Parameter

try {

setAge(-5); // Setting negative age

} catch (IllegalArgumentException e) {

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

}

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

try {

processString(""); // Passing empty string

} catch (IllegalArgumentException e) {

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

}

}

static double calculateCircleArea(int radius) {

if (radius < 0) {

throw new IllegalArgumentException("Radius cannot be negative");

}

return Math.PI \* radius \* radius;

}

enum Month {

JANUARY, FEBRUARY, MARCH, APRIL, MAY, JUNE, JULY, AUGUST, SEPTEMBER, OCTOBER, NOVEMBER, DECEMBER

}

static void printMonth(int monthNumber) {

if (monthNumber < 1 || monthNumber > 12) {

throw new IllegalArgumentException("Invalid month number");

}

System.out.println("Month: " + Month.values()[monthNumber - 1]);

}

static void setAge(int age) {

if (age < 0) {

throw new IllegalArgumentException("Age cannot be negative");

}

System.out.println("Age: " + age);

}

static void processString(String str) {

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

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

}

System.out.println("Processed String: " + str);

}

}

5. class NegativeValueException extends Exception {

public NegativeValueException(String message) {

super(message);

}

}

class InputValidator {

static void validatePositive(int number) throws NegativeValueException {

if (number < 0) {

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

}

}

}

public class Main {

public static void main(String[] args) {

try {

int userInput = -5;

InputValidator.validatePositive(userInput);

} catch (NegativeValueException e) {

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

}

}

}

6. class InvalidDecimalException extends Exception {

public InvalidDecimalException(String message) {

super(message);

}

}

class InputValidator {

static void validateInteger(double number) throws InvalidDecimalException {

if (number % 1 != 0) {

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

}

}

}

public class Main {

public static void main(String[] args) {

try {

double userInput = 3.14;

InputValidator.validateInteger(userInput);

} catch (InvalidDecimalException e) {

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

}

}

}

7. class NonIntegerInputException extends Exception {

public NonIntegerInputException(String message) {

super(message);

}

}

class InputValidator {

static void validateInteger(String input) throws NonIntegerInputException {

try {

Integer.parseInt(input);

} catch (NumberFormatException e) {

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

}

}

}

public class Main {

public static void main(String[] args) {

try {

String userInput = "abc";

InputValidator.validateInteger(userInput);

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

} catch (NonIntegerInputException e) {

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

}

}

}

8. import java.io.IOException;

class MyClass {

void myMethod() throws IOException {

throw new IOException("IOException occurred");

}

}

9. public class ThreadTimingExample {

public static void main(String[] args) {

Thread t1 = new Thread(() -> {

try {

Thread.sleep(2000);

} catch (InterruptedException e) {

e.printStackTrace();

}

});

Thread t2 = new Thread(() -> {

try {

Thread.sleep(3000);

} catch (InterruptedException e) {

e.printStackTrace();

}

});

long startTime = System.currentTimeMillis();

t1.start();

t2.start();

try {

t1.join();

t2.join();

} catch (InterruptedException e) {

e.printStackTrace();

}

long endTime = System.currentTimeMillis();

if (t1.getState() == Thread.State.TIMED\_WAITING && t2.getState() == Thread.State.TIMED\_WAITING) {

System.out.println("Both threads waited for the same duration.");

} else if (t1.getState() == Thread.State.TIMED\_WAITING) {

System.out.println("Thread 1 waited longer.");

} else if (t2.getState() == Thread.State.TIMED\_WAITING) {

System.out.println("Thread 2 waited longer.");

}

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

}

}

public class Main {

public static void main(String[] args) {

try {

MyClass obj = new MyClass();

obj.myMethod();

} catch (IOException e) {

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

}

}

}

10. class MyThread extends Thread {

private long sleepingTime;

private long waitingTime;

public MyThread(String name, long sleepingTime, long waitingTime) {

super(name);

this.sleepingTime = sleepingTime;

this.waitingTime = waitingTime;

}

@Override

public void run() {

try {

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

System.out.println(Thread.currentThread().getName() + " is going to sleep for " + sleepingTime + " milliseconds.");

long startTime = System.currentTimeMillis();

Thread.sleep(sleepingTime);

long endTime = System.currentTimeMillis();

System.out.println(Thread.currentThread().getName() + " woke up after sleeping for " + (endTime - startTime) + " milliseconds.");

synchronized (this) {

System.out.println(Thread.currentThread().getName() + " is going to wait for " + waitingTime + " milliseconds.");

startTime = System.currentTimeMillis();

wait(waitingTime);

endTime = System.currentTimeMillis();

System.out.println(Thread.currentThread().getName() + " waited for " + (endTime - startTime) + " milliseconds.");

}

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

public class Main {

public static void main(String[] args) {

MyThread t1 = new MyThread("Thread 1", 2000, 3000);

MyThread t2 = new MyThread("Thread 2", 3000, 2000);

t1.start();

t2.start();

try {

t1.join();

t2.join();

} catch (InterruptedException e) {

e.printStackTrace();

}

long t1TotalTime = t1.sleepingTime + t1.waitingTime;

long t2TotalTime = t2.sleepingTime + t2.waitingTime;

System.out.println();

if (t1TotalTime > t2TotalTime) {

System.out.println("Thread 1 has longer waiting and sleeping time.");

} else if (t2TotalTime > t1TotalTime) {

System.out.println("Thread 2 has longer waiting and sleeping time.");

} else {

System.out.println("Both threads have equal waiting and sleeping time.");

}

}

}

11. public class JoinThreadsExample {

public static void main(String[] args) {

Thread t1 = new Thread(() -> {

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

if (i % 2 == 0) {

System.out.println("Even number found in Thread 1: " + i);

}

}

});

Thread t2 = new Thread(() -> {

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

if (i % 2 == 0) {

System.out.println("Even number found in Thread 2: " + i);

}

}

});

t1.start();

t2.start();

try {

if (t1.getState() == Thread.State.RUNNABLE && t2.getState() == Thread.State.RUNNABLE) {

t1.join(10000); // T1 waits for 10 seconds

t2.join(5000); // T2 waits for 5 seconds

}

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

12. class MatrixThread extends Thread {

int[][] matrix;

public MatrixThread(int[][] matrix) {

this.matrix = matrix;

}

@Override

public void run() {

System.out.println("Thread " + Thread.currentThread().getId() + " is executing.");

// Perform matrix operations here

}

}

public class MatrixExample {

public static void main(String[] args) {

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();

try {

t1.join();

t2.join();

} catch (InterruptedException e) {

e.printStackTrace();

}

System.out.println("Threads T1 and T2 have completed their operations.");

// Start thread T3 after threads T1 and T2 have finished

// Implement logic for matrix addition here

}

}