**Batch: B-1 Roll No.: 16010122104**

**Experiment / assignment / tutorial No.07**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

|  |
| --- |
| **TITLE : User Defined Exception** |

**AIM:**

Create a user defined exception subclass NumberException with necessary constructor and overridden toString method. Write a program which accepts a number from the user. It throws an object of the NumberException class if the number contains digit 3 otherwise it displays the appropriate message. On printing, the exception object should display an exception name, appropriate message for exception.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Expected OUTCOME of Experiment:**

**CO1:** Understand the features of object oriented programming compared with procedural approach with C++ and Java

**CO4:**Explore the interface, exceptions, multithreading, packages **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

1.Ralph Bravaco , Shai Simoson , “Java Programming From the Group Up” Tata McGraw-Hill.

2.Grady Booch, Object Oriented Analysis and Design.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Pre Lab/ Prior Concepts:**

**Exception handling** in java is a powerful mechanism or technique that allows us to handle runtime errors in a program so that the normal flow of the program can be maintained. All the exceptions occur only at runtime. A syntax error occurs at compile time.

**Exception in Java:**

In general, an exception means a problem or an abnormal condition that stops a computer program from processing information in a normal way.

An exception in java is an object representing an error or an abnormal condition that occurs at runtime execution and interrupts (disrupts) the normal execution flow of the program.

An exception can be identified only at runtime, not at compile time. Therefore, it is also called runtime errors that are thrown as exceptions in Java. They occur while a program is running.

For example:

* If we access an array using an index that is out of bounds, we will get a runtime error named ArrayIndexOutOfBoundsException.
* If we enter a double value while the program is expecting an integer value, we will get a runtime error called InputMismatchException.

When JVM faces these kinds of errors or dividing an integer by zero in a program, it creates an exception object and throws it to inform us that an error has occurred.If the exception object is not caught and handled properly, JVM will display an error message and will terminate the rest of the program abnormally.

If we want to continue the execution of remaining code in the program, we will have to handle exception objects thrown by error conditions and then display a user-friendly message for taking corrective actions. This task is known as exception handling in java.

**Types of Exceptions in Java**

Basically, there are two types of exceptions in java API. They are:

1. Predefined Exceptions (Built-in-Exceptions)

2. Custom (User defined)Exceptions

**Predefined Exceptions:**

Predefined exceptions are those exceptions that are already defined by the Java system. These exceptions are also called built-in-exceptions.Java API supports exception handling by providing the number of predefined exceptions. These predefined exceptions are represented by classes in java.

When a predefined exception occurs, JVM (Java runtime system) creates an object of predefined exception class. All exceptions are derived from java.lang.Throwable class but not all exception classes are defined in the same package. All the predefined exceptions supported by java are organized as subclasses in a hierarchy under the Throwable class.

All the predefined exceptions are further divided into two groups:

1. Checked Exceptions: Checked exceptions are those exceptions that are checked by the java compiler itself at compilation time and are not under runtime exception class hierarchy. If a method throws a checked exception in a program, the method must either handle the exception or pass it to a caller method.

2. Unchecked Exceptions: Unchecked exceptions in Java are those exceptions that are checked by JVM, not by java compiler. They occur during the runtime of a program. All exceptions under the runtime exception class are called unchecked exceptions or runtime exceptions in Java.

**Custom exceptions:**

Custom exceptions are those exceptions that are created by users or programmers according to their own needs. The custom exceptions are also called user-defined exceptions that are created by extending the exception class.

So, Java provides the liberty to programmers to throw and handle exceptions while dealing with functional requirements of problems they are solving.

**Exception Handling Mechanism using Try-Catch block:**

The general syntax of try-catch block (exception handling block) is as follows:

**Syntax:**

try

{

// A block of code; // generates an exception

}

catch(exception\_class var)

{

// Code to be executed when an exception is thrown.

}

**Example:**

public class TryCatchEx

{

public static void main(String[] args)

{

System.out.println("11");

System.out.println("Before divide");

int x = 1/0;

System.out.println("After divide");

System.out.println("22");

}

}

Output**:**

11

Before divide

Exception in thread "main" java.lang.ArithmeticException: / by zero

**Class Diagram:**

+---------------------------------+

| Exception |

+---------------------------------+

| -message: String |

+---------------------------------+

| +Exception(message: String) |

| +toString(): String |

+---------------------------------+

+---------------------------------+

| NumberException |

+---------------------------------+

| +NumberException(message: String)|

+---------------------------------+

+---------------------------------+

| Main |

+---------------------------------+

| +main(args: String[]): void |

| +containsDigit3(number: int): boolean |

+---------------------------------+

**Algorithm:**

Algorithm: CheckForDigitThree

Step 1: Start

Step 2: Create a class Exception with a private attribute message of type String.

Step 3: Create a constructor Exception(message) that initializes the message attribute.

Step 4: Create a public method toString() in the Exception class that returns the message attribute as a string.

Step 5: Create a class NumberException that inherits from Exception.

Step 6: Create a constructor NumberException(message) that calls the constructor of the parent class Exception with the provided message.

Step 7: Create a class Main for the main program logic.

Step 8: In the Main class:

a. Define a method containsDigit3(number) that takes an integer number as input and returns true if the number contains the digit '3'; otherwise, return false.

b. Implement the main method:

i. Display a message asking the user to enter a number.

ii. Read the user input as an integer and store it in a variable 'userNumber'.

iii. Use the containsDigit3 method to check if 'userNumber' contains the digit '3':

- If it contains '3', create an instance of NumberException with an appropriate error message and throw the exception.

- If it does not contain '3', display a message indicating that the number is valid.

iv. Handle exceptions:

- If a NumberException is caught, print the exception message.

- If other exceptions occur, print a generic error message.

Step 9: End

**Implementation details :**

import java.util.Scanner;

// Custom exception class

class NumberException extends Exception {

    public NumberException(String message) {

        super(message);

    }

    @Override

    public String toString() {

        return "NumberException: " + getMessage();

    }

}

public class Main {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        try {

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

            int number = scanner.nextInt();

            // Check if the number contains the digit 3

            if (containsDigit3(number)) {

                throw new NumberException("Number contains the digit 3.");

            } else {

                System.out.println("The number does not contain the digit 3.");

            }

        } catch (NumberException e) {

            // Handle the custom exception

            System.out.println(e);

        } catch (Exception e) {

            // Handle other exceptions

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

        } finally {

            scanner.close();

        }

    }

    // Helper method to check if a number contains the digit 3

    private static boolean containsDigit3(int number) {

        while (number > 0) {

            int digit = number % 10;

            if (digit == 3) {

                return true;

            }

            number /= 10;

        }

        return false;

    }

}

**Output:**

A black background with colorful text

Description automatically generated with medium confidence

**Conclusion:**

In this Java program, a custom exception called `NumberException` is created with overridden `toString` method, and it is used to check if a user-provided number contains the digit '3,' throwing the exception with an appropriate message if the condition is met, otherwise displaying a valid message, and the exception object can display both its name and an appropriate exception message when printed.

**Date: 09/10/2023 Signature of faculty in-charge**