Batch: B2 Roll No.: 16010124107
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:

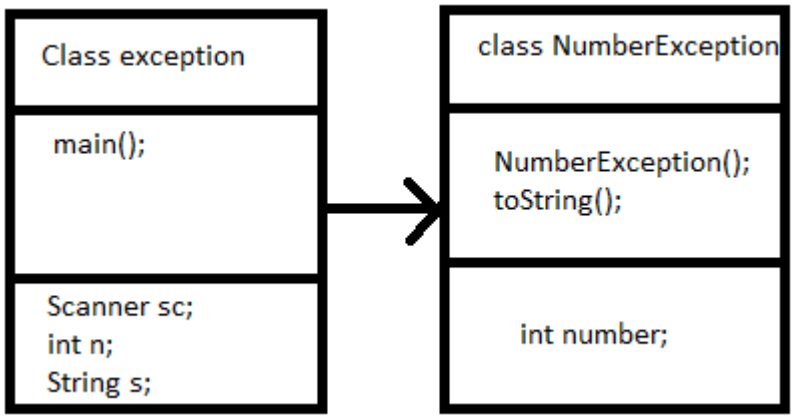
```
public class TryCatchEx
{
public static void main(String[] args)
{
 System.out.println("11");
 System.out.println("Before divide");
 int x = 1/0;
 System.out.println("After divide");
 System.out.println("22");
          }
}
```

Output:

```
    11
```

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

Class Diagram:



Algorithm:

1. Start
2. Initialize class exception
3. Declare a main class, declare the scanner class
4. Input a number
5. Try the exception, send through function toString(); if it contains 3, throw exception else print statement.
6. Inside the catch block, call the exception object and print the appropriate message.
7. Stop

Implementation details:

```
import java.util.*;


class NumberException extends Exception {
    public int number;


    NumberException(int number) {
```

```
        this.number = number;

    }


    @Override

    public String toString() {

         return "NumberException: The number " + number + " contains the
digit 3.";

    }

}


public class exception {

    public static void main(String args[]) {

        Scanner sc = new Scanner(System.in);

        System.out.println("Enter the number: ");

        int n = sc.nextInt();


        try {

            String s = Integer.toString(n);

            if (s.contains("3")) {

                throw new NumberException(n);

            } else {

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

            }
```

```
        } catch (NumberException e) {

            System.out.println(e);

        }

    }

}
```

Output:

```
Enter the number:
456
The number does not contain digit 3.
PS C:\Users\STUDENT\Desktop\java_23> java exception
Enter the number:
361
NumberException: The number 361 contains the digit 3.
PS C:\Users\STUDENT\Desktop\java_23>
```

Conclusion:

Exception handling in Java is a powerful mechanism that allows programs to deal with unexpected situations in a structured and controlled way. By separating error-handling code from regular business logic, it improves program readability, reliability, and maintainability.

Date: 29/09/2025 **Signature of faculty in-charge**

Post Lab Descriptive Questions

1. Compare throw and throws.

Throw	Throws
Used to explicitly throw a single exception inside a method or block of code.	Used in the method declaration to specify which exceptions a method might throw.

Appears inside a method or block.	Appears in the method signature.
Can throw only one exception at a time.	Can declare multiple exceptions (comma-separated).
throw new ExceptionType("Message");	returnType methodName() throws Exception1, Exception2 { ... }
Actually creates and throws the exception object.	Only declares the possibility of exceptions, doesn't throw itself.

2. **Write program to implement following problem statement:**

Create a User-Defined Exception:

Define a custom exception class named InsufficientFundsException that extends the built-in Exception class.This exception should be thrown when a withdrawal request exceeds the available balance in the bank account.

Bank Account Class:

Create a class named BankAccount with the following attributes and methods:

private double balance (the current balance of the account)

A constructor to initialize the balance.

A method deposit(double amount) to add funds to the account.

A method withdraw(double amount) that throws the InsufficientFundsException if the amount to be withdrawn exceeds the current balance. Otherwise, it should deduct the amount from the balance.

A method getBalance() to return the current balance of the account.

Main Class:

In the main method of your application, demonstrate how to use the BankAccount class and handle the InsufficientFundsException.

Create a BankAccount object, perform a few deposits, and attempt to withdraw an amount that might cause an exception. Catch the InsufficientFundsException and print an appropriate error message.

Implementation Details:

```
class InsufficientFundsException extends Exception {

    private double amount;


    InsufficientFundsException(double amount) {

        this.amount = amount;

    }


    @Override

    public String toString() {

         return "InsufficientFundsException: Withdrawal of " + amount + "
exceeds available balance.";

    }

}


class BankAccount {

    private double balance;


    BankAccount(double balance) {

        this.balance = balance;

    }


    public void deposit(double amount) {
```

```
        balance += amount;

         System.out.println("Deposited: " + amount + ", New Balance: " +
balance);

    }


    public void withdraw(double amount) throws InsufficientFundsException
{

        if (amount > balance) {

            throw new InsufficientFundsException(amount);

        } else {

            balance -= amount;

                System.out.println("Withdrawn: " + amount + ", Remaining
Balance: " + balance);

        }

    }


    public double getBalance() {

        return balance;

    }

}


public class Bank {

    public static void main(String[] args) {

        BankAccount account = new BankAccount(5000);
```

```
        account.deposit(2000);

        account.deposit(1500);



        try {

            account.withdraw(3000);

            account.withdraw(5000);

        } catch (InsufficientFundsException e) {

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

        }



        System.out.println("Final Balance: " + account.getBalance());

    }

}
```

Output:

```
PS C:\Users\STUDENT\Desktop\java_23> java Bank
Deposited: 2000.0, New Balance: 7000.0
Deposited: 1500.0, New Balance: 8500.0
Withdrawn: 3000.0, Remaining Balance: 5500.0
Error: InsufficientFundsException: Withdrawal of 10000.0 exceeds available balance.
Final Balance: 5500.0
PS C:\Users\STUDENT\Desktop\java_23>
```