Batch: A1 **Roll No.: 16010123012**

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 Scanner and ask user to enter a number
3. Store the user's input as a string
4. Try-Catch Block:

   Try:

   Call the checkNumber method, passing the user input.

   If no exception is thrown, print a message indicating that the number does not contain the digit '3'.

   Catch:

   If a NumberException is thrown, print the exception message.

   Define the checkNumber method
5. If the string number contains the character '3'

   Throw a NumberException with the message "The number contains the digit 3".
6. If it does not contain '3', the method completes without throwing an exception
7. Close Scanner
8. End

Implementation details:

```
import java.util.*;

class  NumberException extends Exception{
    public NumberException(String message) {
        super(message);
    }
    @Override
    public String toString() {
        return "NumberException: " + getMessage();
    }
}

public class NumberCheck {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
```

```
        System.out.print("Enter a number: ");
        String input = sc.nextLine();

        try {
            checkNumber(input);
            System.out.println("The number does not contain the digit
3");
        } catch (NumberException e) {
            System.out.println(e);
        }
        sc.close();
    }

    public static void checkNumber(String number) throws NumberException
{
        if (number.contains("3")) {
            throw new NumberException("The number contains the digit 3");
        }
    }
}
```

Output:

```
C:\Users\KJSCE\Desktop\Java>javac NumberCheck.java
Picked up JAVA_TOOL_OPTIONS: -Dstdout.encoding=UTF-8 -Dstderr.encoding=UTF-8

C:\Users\KJSCE\Desktop\Java>java NumberCheck
Picked up JAVA_TOOL_OPTIONS: -Dstdout.encoding=UTF-8 -Dstderr.encoding=UTF-8
Enter a number: 3
NumberException: The number contains the digit 3
```

```
C:\Users\KJSCE\Desktop\Java>javac NumberCheck.java
Picked up JAVA_TOOL_OPTIONS: -Dstdout.encoding=UTF-8 -Dstderr.encoding=UTF-8

C:\Users\KJSCE\Desktop\Java>java NumberCheck
Picked up JAVA_TOOL_OPTIONS: -Dstdout.encoding=UTF-8 -Dstderr.encoding=UTF-8
Enter a number: 462975
The number does not contain the digit 3
```

```
C:\Users\KJSCE\Desktop\Java>javac NumberCheck.java
Picked up JAVA_TOOL_OPTIONS: -Dstdout.encoding=UTF-8 -Dstderr.encoding=UTF-8

C:\Users\KJSCE\Desktop\Java>java NumberCheck
Picked up JAVA_TOOL_OPTIONS: -Dstdout.encoding=UTF-8 -Dstderr.encoding=UTF-8
Enter a number: 123
NumberException: The number contains the digit 3
```

Conclusion:

We have successfully performed this experiment and learned about exceptions

Date: 08 / 10 / 24 **Signature of faculty in-charge**

Post Lab Descriptive Questions

1. Compare throw and throws.

Key Points	**Throw**	**Throws**
Point of Usage	The **throw** keyword is used inside a function. It is used when it is required to throw an Exception logically.	The **throws** keyword is used in the function signature. It is used when the function has some statements that can lead to exceptions.
Exceptions Thrown	The **throw** keyword is used to throw an exception explicitly. It can throw only one exception at a time.	The **throws** keyword can be used to declare multiple exceptions, separated by a comma. Whichever exception occurs, if matched with the declared ones, is thrown automatically then.
Syntax	Syntax of **throw** keyword includes the instance of the Exception to be thrown. Syntax wise throw keyword is followed by the instance variable.	Syntax of **throws** keyword includes the class names of the Exceptions to be thrown. Syntax wise throws keyword is followed by exception class names.
Propagation of Exceptions	**throw** keyword cannot propagate checked exceptions. It is only used to propagate the unchecked Exceptions that are not checked using the throws keyword.	**throws** keyword is used to propagate the checked Exceptions only.

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.

```
import java.util.*;

class InsufficientFundsException extends Exception {
    public InsufficientFundsException(String message) {
        super(message);
    }
}

class BankAccount {
    private double balance;

    public BankAccount(double balance) {
        this.balance = balance;
    }

    public void deposit(double amount) {
        if (amount > 0) {
            balance += amount;
            System.out.println("Deposited amount: " + amount);
        } else {
            System.out.println("Deposit amount must be positive.");
        }
    }

    public void withdraw(double amount) throws InsufficientFundsException
{
        if (amount > balance) {
            throw new InsufficientFundsException("Withdrawal amount
exceeds available balance.");
        } else {
            balance -= amount;
            System.out.println("Withdrew: " + amount);
        }
    }
```

```
    public double getBalance() {
        return balance;
    }
}

public class BankApp {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        BankAccount account = new BankAccount(470000.00);
        account.deposit(37600.00);
        System.out.println("Current Balance: " + account.getBalance());

        System.out.print("Enter amount to withdraw: ");
        double amount = sc.nextDouble();

        try {
            account.withdraw(amount);
            System.out.println("Current Balance after withdrawal: " +
account.getBalance());
        } catch (InsufficientFundsException e) {
            System.out.println("Error: " + e.getMessage());
        } finally {
            sc.close();
        }
    }
}
```

```
PS D:\KJSCE\SY\OOPS\Program> javac BankApp.java
PS D:\KJSCE\SY\OOPS\Program> java BankApp
Deposited amount: 37600.0
Current Balance: 507600.0
Enter amount to withdraw: 470000
Withdrew: 470000.0
Current Balance after withdrawal: 37600.0
PS D:\KJSCE\SY\OOPS\Program> javac BankApp.java
PS D:\KJSCE\SY\OOPS\Program> java BankApp
Deposited amount: 37600.0
Current Balance: 507600.0
Enter amount to withdraw: 10000000
Error: Withdrawal amount exceeds available balance.
```