Batch:B4 Roll No.:16010122221
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

Algorithm:

Step 1: Create a class that extends the class Exception and call the constructor of that class.

Step 2: In the constructor initialize the class variables.

Step 3: In the main method of class Exp7 input the time from the user and check if they are valid in the try block.

Step 4: If the time is invalid then throws an exception .In the catch block print the exception name and the message that needs to be printed along with it.

Step 5: If the time entered is valid then display the total seconds.

Implementation details :

```
import java.util.Scanner;


class TimeException extends Exception {
    String msg;


    public TimeException(String s) {
        this.msg = s;
    }


    public String toString() {
        return (msg);
    }
}


public class Exp7 {
    public static void main(String args[]) {
        Scanner sc = new Scanner(System.in);
        try {
            System.out.print("Enter minutes: ");
            int min = sc.nextInt();
            System.out.print("Enter seconds: ");
            int sec = sc.nextInt();
            if (sec > 60)
```

```
                throw new TimeException("seconds can't be more than 60
seconds");

            else {

                sec = 60 * min + sec;

                System.out.println("Total Seconds is " + sec + " sec");

            }

        } catch (TimeException e) {

            System.out.print("TimeException Error: ");

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

        }

    }

}
```

Output:

```
Enter minutes: 2
Enter seconds: 20
Total Seconds is 140 sec
```

```
Enter minutes: 4
Enter seconds: 10
Total Seconds is 250 sec
```

Conclusion:

User defined exceptions were understood and implemented successfully.

Date: _____ **Signature of faculty in-charge**

Post Lab Descriptive Questions

1. Compare throw and throws.

throw	throws
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.
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 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.
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. Explain how to create a user defined exception and explicitly throw an exception in a program with a simple example.

Java user-defined exception is a custom exception created and throws that exception using the keyword ‘throw’. It is done by extending a class ‘Exception’. An exception is a
problem that arises during the execution of the program. In Object-Oriented

Programming language, Java provides a powerful mechanism to handle such exceptions. Java allows to create own exception class, which provides own exception class implementation. Such exceptions are called user-defined exceptions or custom exceptions.

Example:

```
class Example {
    public static void main(String args[]) {
    try {
    throw new NewException(100);
    } catch (NewException e) {
    System.out.println(e);
    }
    }
}
class NewException extends Exception {
int a;
NewException(int b) {
a = b;
}
public String toString() {
return ("Status code = " + a);
}
}
```

3. Suppose the statement2 causes an exception in following try-catch block:

```
try {
        statement1;
        statement2;
        statement3;
}
catch(Exception1 e1) {
}
        catch(Exception2 e2){
}
```

statement4;

Answer the following questions:
● Will statement3 be executed?
NO, the statement 3 will not be executed.
● If the exception is not caught, will statement4 be executed?
NO, statement 4 will not be executed.
● If the exception is caught in the catch block, will statement4 be executed?
YES, if the exception is caught in the catch block, statement 4 will be executed.
● If the exception is passed to the caller, will the statement4 be executed?
YES, if the exception is passed to the caller, statement 4 will be executed.

4. Explain finally block with the help of an example.

The finally block in java is used to put important codes such as clean up code e.g. closing the file or closing the connection. The finally block executes whether exception rise or not and whether exception handled or not. A finally contains all the crucial statements regardless of the exception occurs or not.

```
class Test {
public static void main(String args[]) {
try {
int data = 25 / 5;
System.out.println(data);
} catch (NullPointerException e) {
System.out.println(e);
} finally {
System.out.println("finally block is always
executed");
}
System.out.println("rest of the code...");
  }
}
```