Course Name:	**Object Oriented Programming Methodology**	**Semester:**	**III**
Date of Performance:	**___30 / __09_ /____23____**	**Batch No:**	**B2**
Faculty Name:	**Prof. Kiran Thale**	**Roll No:**	**16010122151**
Faculty Sign & Date:		**Grade/Marks:**	**___/25**

Experiment No: 7
Title: User Defined Exception

Aim and Objective of the Experiment:
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.

COs to be achieved:
CO1: Understand the features of object oriented programming compared with procedural approach with C++ and Java.
CO2: Explore arrays, vectors, classes and objects in C++ and Java.
CO3: Implement scenarios using object oriented concepts (Drawing class diagram, relationship between classes, sequence diagram)
CO4: Explore the interface, exceptions, multithreading, packages

Tools used:
JDK, VScode / Eclipse

Theory:
Pre Lab/ Prior Concepts:
Exception handling in java is a powerful mechanism or technique that allows us to
Define Class, Methods and Object.
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.

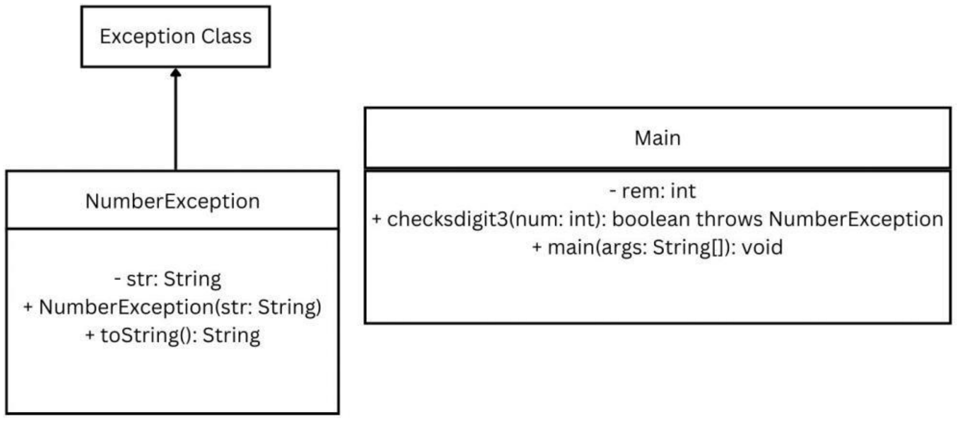
Exception Handling Mechanism using Try-Catch block:
The general syntax of try-catch block (exception handling block) is as follows:
Syntax:
```
try
{
  // A block of code; // generates an exception
}
catch(exception_class var)
{
  // Code to be executed when an exception is thrown.
}
```

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:

Draw the class diagram for each of the problem statements


Exception Class
NumberException
- str: String
+ NumberException(str: String)
+ toString(): String
Main
- rem: int
+ checksdigit3(num: int): boolean throws NumberException
+ main(args: String[]): void

Algorithm:-

1. Start

2. Define a custom exception class NumberException that extends Exception:-

- Create a class NumberException that extends the Exception class.
- Add a constructor that takes a string message as an argument and passes it to the superclass constructor.
- Override the toString() method to return a custom error message.

3. In the Main class, define a static variable rem to store the remainder when a number is divided by 10.

4. Define a static method checkDigit3(int num) that throws NumberException:

- Create a static method checkDigit3 that takes an integer num as an argument and throws a NumberException.
- Inside the method, initialize a variable rem to 0.
- Use a loop to iterate through the digits of the number:-

➢ While num is greater than 0:

❖ Calculate the remainder by taking num % 10 and store it in rem.

❖ If rem is equal to 3, throw a new NumberException with a message indicating that the number contains the digit 3.

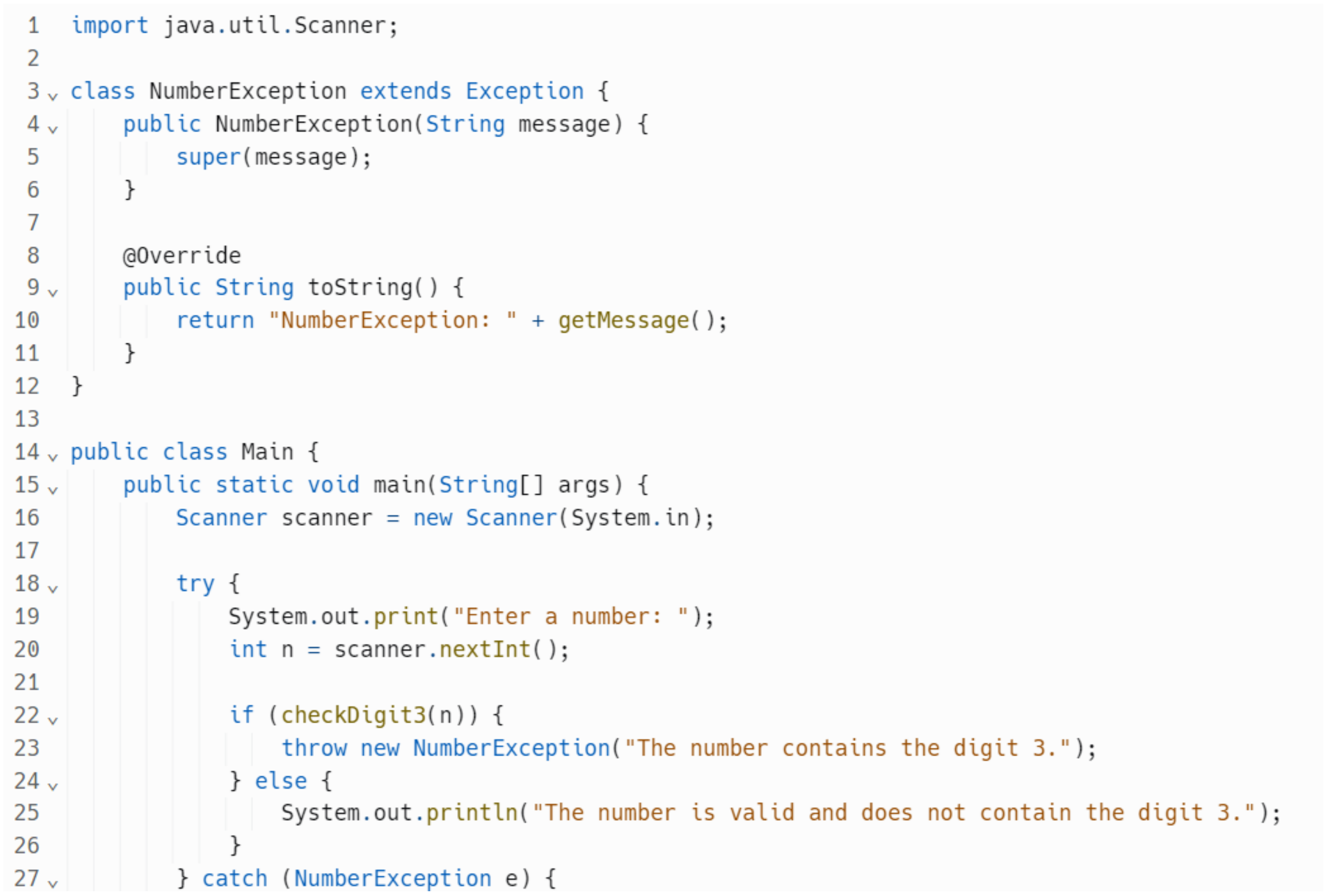
- ❖ Update num by performing integer division num / 10 to check the next digit.
- If the loop completes without finding a 3, return indicating the number is valid.

5. In the main method:-

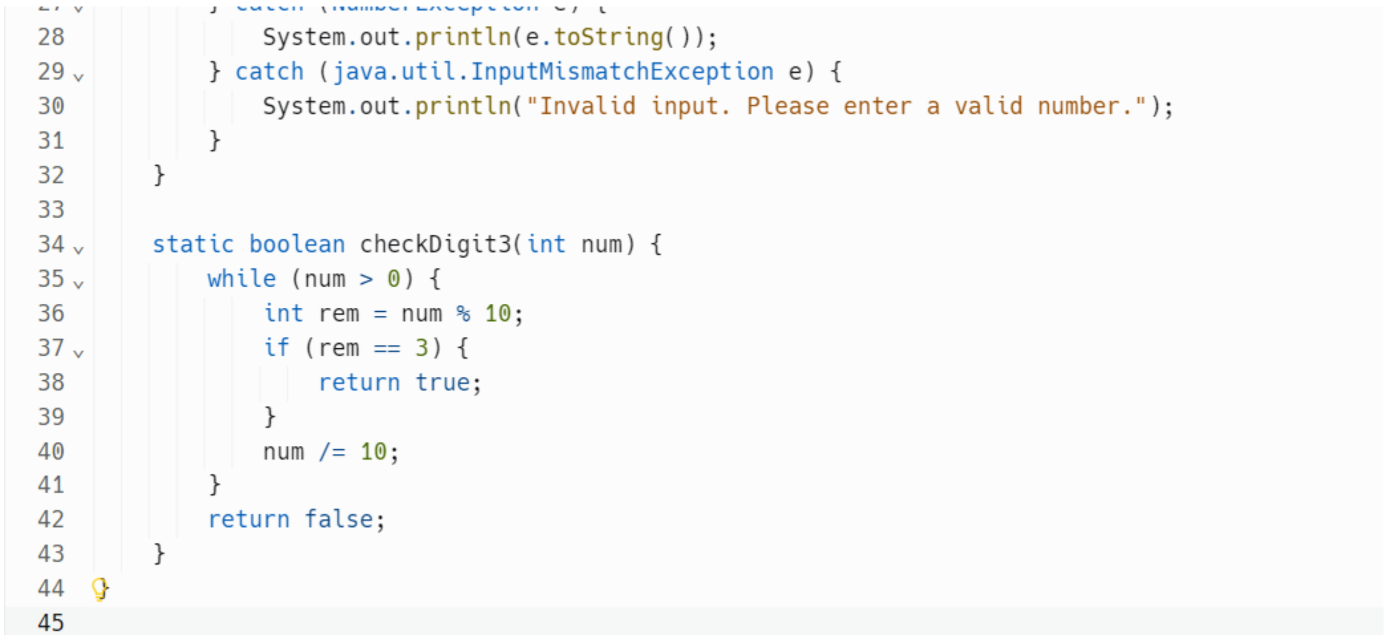
- Create a Scanner object to read input from the user.
- Prompt the user to enter a number and store it in a variable n.
- Call the checkDigit3(n) method in a try block.
- If an exception is thrown:
- Catch the NumberException and print its message using its toString() method.
- If no exception is thrown:
- Print a message indicating that the number is valid and does not contain the digit 3.

6. End.

Code:-

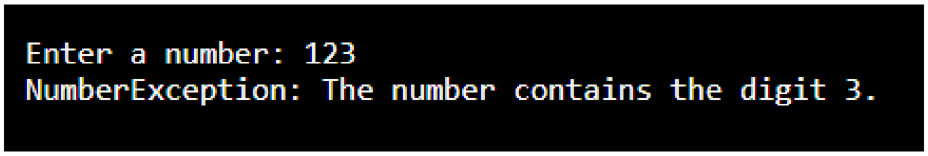


```java
import java.util.Scanner;

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 n = scanner.nextInt();

            if (checkDigit3(n)) {
                throw new NumberException("The number contains the digit 3.");
            } else {
                System.out.println("The number is valid and does not contain the digit 3.");
            }
        } catch (NumberException e) {
```

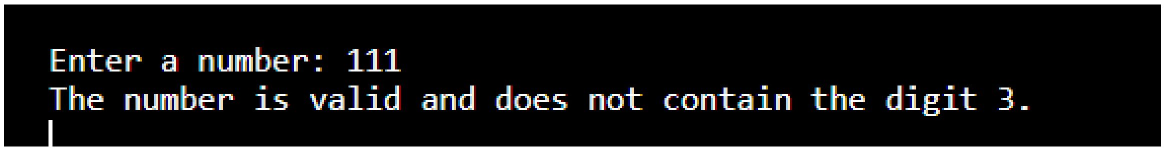


```
        } catch (NumberException e) {
            System.out.println(e.toString());
        } catch (java.util.InputMismatchException e) {
            System.out.println("Invalid input. Please enter a valid number.");
        }
    }

    static boolean checkDigit3(int num) {
        while (num > 0) {
            int rem = num % 10;
            if (rem == 3) {
                return true;
            }
            num /= 10;
        }
        return false;
    }
}
```

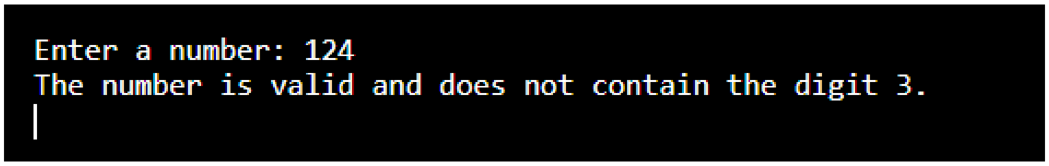
Output:-



```
Enter a number: 123
NumberException: The number contains the digit 3.
```



```
Enter a number: 111
The number is valid and does not contain the digit 3.
```



```
Enter a number: 124
The number is valid and does not contain the digit 3.
```

Post Lab Subjective/Objective type Questions:

Q1) Compare throw and throws.
Ans)
Usage:-

- throw is used within a function or a block of code when you want to throw an exception explicitly and handle it within the same function or in a calling function.
- throws is used in the function signature to indicate that the function may throw exceptions. It's a way to declare the exceptions that can be thrown by the function.

Exceptions Thrown:-

- throw is used to throw an exception explicitly. It can throw only one exception at a time, and it's used for raising custom or predefined exceptions.
- throws can be used to declare multiple exceptions in the function signature, separated by commas. It indicates the types of exceptions that might be thrown by the function during its execution.

Syntax:-

- The throw keyword is followed by an instance of the Exception class that you want to throw.
- The throws keyword is followed by the class names of the exceptions that the function may throw.

Propagation of Exceptions:-

- throw is typically used for propagating unchecked exceptions (subclasses of RuntimeException). These exceptions don't need to be declared using throws in the method signature.
- throws is used to propagate checked exceptions. When a method declares that it throws a checked exception, it indicates that the caller of the method should handle or declare that exception.
- Your explanation provides a clear distinction between these two keywords and their use in handling exceptions in Java. This knowledge is valuable for understanding how to work with exceptions effectively in your Java programs.

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

In Java, you can create a user-defined exception by extending the Exception class.

```
class MyException extends Exception
```

```
{


    public MyException(String message)
   {
        super(message);
     }
}

class Test
{
     static void validate(int age) throws MyException
     {
          if (age < 18)
          {
               throw new MyException("Not valid");
          }
          else
         {
               System.out.println("Welcome to vote");
        }
    }

    public static void main(String args[])
   {
        try
       {
            validate(13);
        }
        catch (MyException m)
        {
            System.out.println("Exception occured: " + m);
        }
    }
}
```

In this example, we’ve created a user-defined exception MyException. The validate method throws MyException when the age is less than 18. In the main method, we handle this exception with a try-catch block.

1. 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?

Yes, statement3 will be executed if no exceptions are thrown in statement1 or statement2. If an exception is thrown in either statement1 or statement2, the execution will immediately jump to the appropriate catch block, and statement3 will not be executed.

- If the exception is not caught, will statement4 be executed?

No, if an exception is thrown and not caught in the try-catch block, the program will terminate immediately and statement4 will not be executed.

- If the exception is caught in the catch block, will statement4 be executed?

Yes, if an exception is thrown and it's caught in one of the catch blocks, after executing the catch block, the program will continue with statement4.

- If the exception is passed to the caller, will the statement4 be executed?

No, if an exception is thrown and passed to the caller (i.e., it's not caught in this method), the program will immediately return to the caller method and statement4 will not be executed.

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

In Java, a finally block is a key tool for ensuring that certain essential parts of your code are always executed before a method returns, regardless of whether an exception was thrown.
The finally block follows a try or try-catch block and contains cleanup code that is always executed when the try block exits. This is true even if an unexpected exception occurs.

```
public class Finally
{
    public static void main(String[] args)
   {
        try
       {
            int divideByZero = 5 / 0;
            System.out.println("Rest of try block");
        }
        catch (ArithmeticException e)
       {
            System.out.println("ArithmeticException => " +
e.getMessage());
        }
        finally
       {
            System.out.println("This is the finally block");
        }
        System.out.println("Outside the try-catch-finally block");
    }
}
```

Conclusion:

The experiment demonstrates the use of custom exceptions in Java. The NumberException class is created to handle a specific scenario where a number contains the digit 3. The Main class includes a method that checks for this condition and throws the custom exception if it's met. This showcases how Java's exception handling mechanism can be extended and customized for specific application requirements.

Signature of faculty in-charge with Date: