Batch: E2 **Roll No.: 16010123325**

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.
}
```

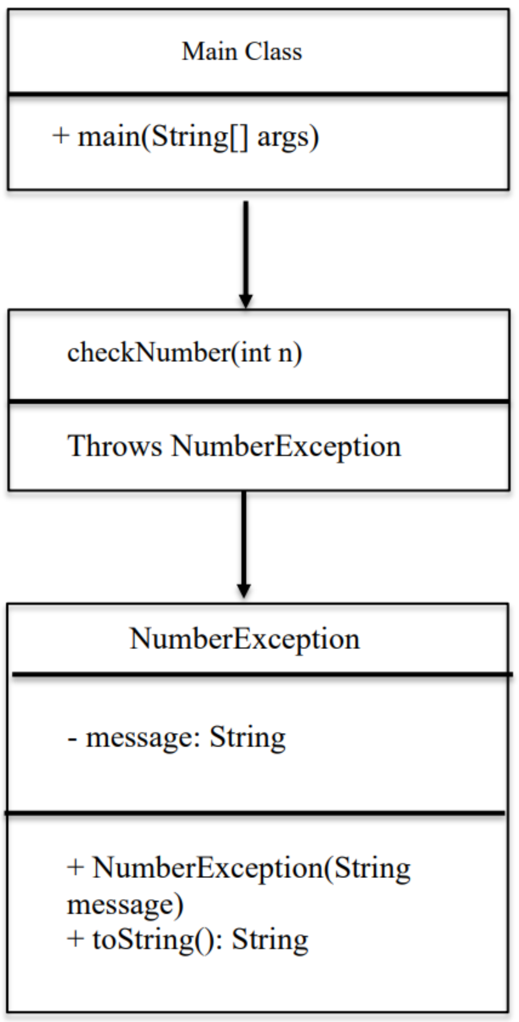
Example:
```
public class TryCatchEx
{
public static void main(String[] args)
{
 System.out.println("11");
 System.out.println("Before divide");
 int x = 1/0;
 System.out.println("After divide");
 System.out.println("22");
```

```
        }
}
```

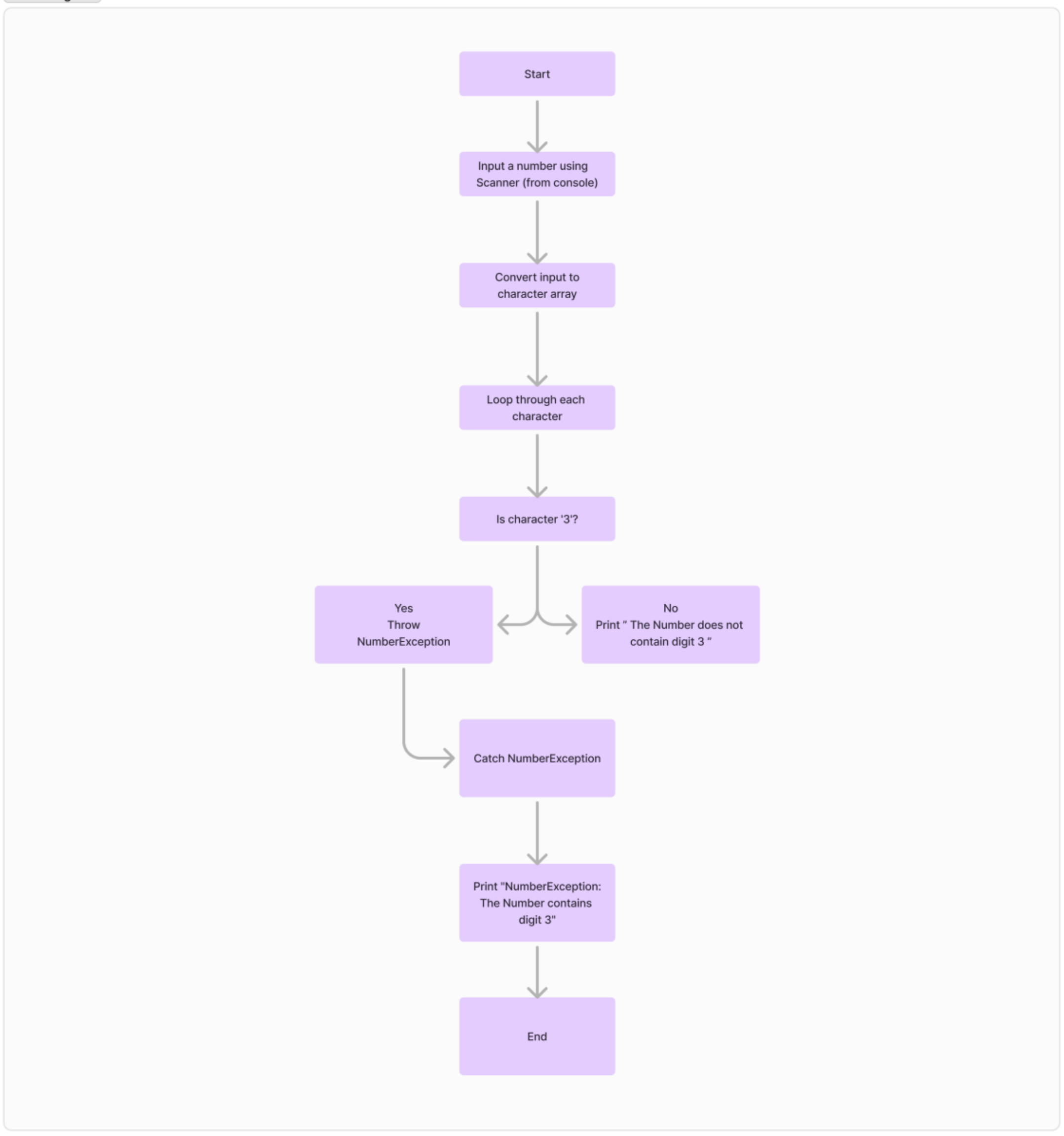
Output:

```
11
Before divide
Exception in thread "main" java.lang.ArithmeticException: / by zero
```

Class Diagram:



Class Diagram


Start
Input a number using Scanner (from console)
Convert input to character array
Loop through each character
Is character '3'?
Yes
Throw NumberException
No
Print " The Number does not contain digit 3 "
Catch NumberException
Print "NumberException: The Number contains digit 3"
End

Algorithm:

Step 1: Initialize

- Create a Scanner object to read user input
- Prompt the user to enter a number

Step 2: Read Input

- Read the user's input using the Scanner object
- Store the input as a String

Step 3: Check for Digit 3

- Convert the input String to a character array
- Iterate through each character in the array
- Check if the character is equal to '3'
- If '3' is found, set a boolean flag isPresent to true and break the loop

Step 4: Handle Exception

- If isPresent is true, throw a NumberException with a custom error message
- If isPresent is false, print a success message indicating that the number does not contain the digit 3

Step 5: Catch Exception

- Catch the NumberException thrown in Step 4
- Print the error message using the toString() method of the NumberException class

Step 6: Terminate

- The program terminates after handling the exception or printing the success message.

Implementation details :

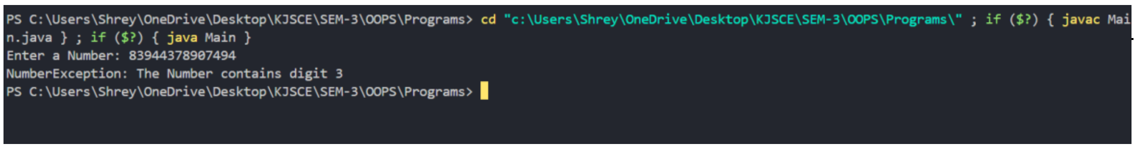
In NumberException.java file

```java
public class NumberException extends Exception {
    public NumberException(String message) {
        super(message);
    }

    @Override
    public String toString() {
        return "NumberException: " + getMessage();
    }
}
```

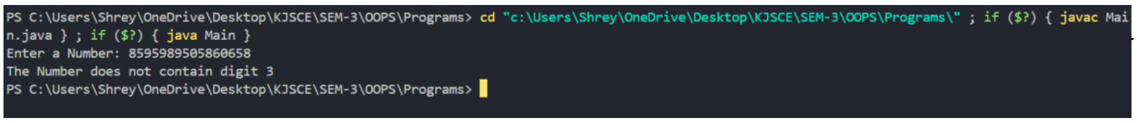
In Main.java file

```java
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {
        try (Scanner scanner = new Scanner(System.in)) {
            System.out.print("Enter a Number: ");
            String input = scanner.next();
            boolean isPresent = false;
            for (char c : input.toCharArray()) {
                if (c == '3') {
                    isPresent = true;
                    break;
                }
            }
            if (isPresent) {
                throw new NumberException("The Number contains digit 3");
            }
            else {
                System.out.println("The Number does not contain digit
3");
            }
        }
        catch (NumberException e) {
            System.out.println(e.toString());
        }
    }
}
```

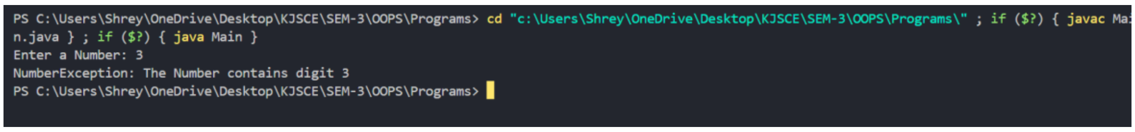
Output:



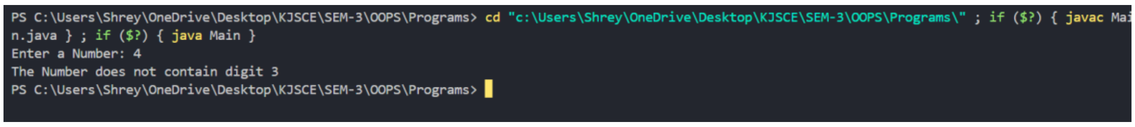
```
PS C:\Users\Shrey\OneDrive\Desktop\KJSCE\SEM-3\OOPS\Programs> cd "c:\Users\Shrey\OneDrive\Desktop\KJSCE\SEM-3\OOPS\Programs\" ; if ($?) { javac Mai
n.java } ; if ($?) { java Main }
Enter a Number: 83944378907494
NumberException: The Number contains digit 3
PS C:\Users\Shrey\OneDrive\Desktop\KJSCE\SEM-3\OOPS\Programs> 
```



```
PS C:\Users\Shrey\OneDrive\Desktop\KJSCE\SEM-3\OOPS\Programs> cd "c:\Users\Shrey\OneDrive\Desktop\KJSCE\SEM-3\OOPS\Programs\" ; if ($?) { javac Mai
n.java } ; if ($?) { java Main }
Enter a Number: 8595989505860658
The Number does not contain digit 3
PS C:\Users\Shrey\OneDrive\Desktop\KJSCE\SEM-3\OOPS\Programs> 
```



```
PS C:\Users\Shrey\OneDrive\Desktop\KJSCE\SEM-3\OOPS\Programs> cd "c:\Users\Shrey\OneDrive\Desktop\KJSCE\SEM-3\OOPS\Programs\" ; if ($?) { javac Mai
n.java } ; if ($?) { java Main }
Enter a Number: 3
NumberException: The Number contains digit 3
PS C:\Users\Shrey\OneDrive\Desktop\KJSCE\SEM-3\OOPS\Programs> 
```



```
PS C:\Users\Shrey\OneDrive\Desktop\KJSCE\SEM-3\OOPS\Programs> cd "c:\Users\Shrey\OneDrive\Desktop\KJSCE\SEM-3\OOPS\Programs\" ; if ($?) { javac Mai
n.java } ; if ($?) { java Main }
Enter a Number: 4
The Number does not contain digit 3
PS C:\Users\Shrey\OneDrive\Desktop\KJSCE\SEM-3\OOPS\Programs> 
```

Conclusion:

The **NumberException** class is implemented with a custom constructor and overridden toString method, and a main program is written to throw this exception when a user-input number contains the digit 3.

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

Post Lab Descriptive Questions

1. Compare throw and throws.

throw

- The throw keyword is used to explicitly throw an exception from a method or block of code.
- It is used to throw an instance of an exception class.
- The general syntax is throw new ExceptionType("Error Message");
- It is used within a method or block of code to throw an exception.

throws

- The throws keyword is used to declare that a method may throw an exception.
- It is used to specify the exceptions that a method can throw.
- The general syntax is methodName() throws ExceptionType1, ExceptionType2, ...
- It is used in the method signature to declare the exceptions that a method can throw.

In conclusion, throw is used to explicitly throw an exception, while throws is used to declare that a method may throw an exception.

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.

Code:

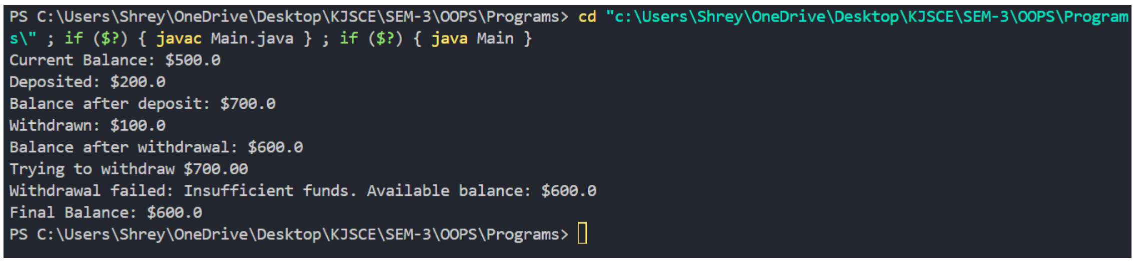
InsufficientFundsException.java

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

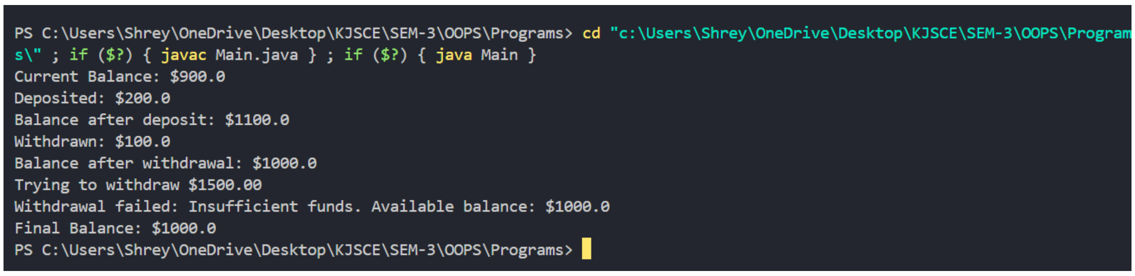
Main.java

```java
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);
        } else {
            System.out.println("Invalid deposit amount");
        }
    }

    public void withdraw(double amount) throws InsufficientFundsException
{
        if (amount > balance) {
            throw new InsufficientFundsException("Withdrawal failed:
Insufficient funds. Available balance: $" + balance);
        } else {
            balance -= amount;
            System.out.println("Withdrawn: $" + amount);
        }
    }

    public double getBalance() {
        return balance;
    }
}

public class Main {
    public static void main(String[] args) {
        BankAccount account = new BankAccount(500.00);

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

        account.deposit(200.00);
        System.out.println("Balance after deposit: $" +
account.getBalance());

        try {
            account.withdraw(100.00);
            System.out.println("Balance after withdrawal: $" +
account.getBalance());
        } catch (InsufficientFundsException e) {
            System.out.println(e.getMessage());
        }

        try {
            System.out.println("Trying to withdraw $700.00");
            account.withdraw(700.00);
        } catch (InsufficientFundsException e) {
            System.out.println(e.getMessage());
        }

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

Output:



```
PS C:\Users\Shrey\OneDrive\Desktop\KJSCE\SEM-3\OOPS\Programs> cd "c:\Users\Shrey\OneDrive\Desktop\KJSCE\SEM-3\OOPS\Program
s\" ; if ($?) { javac Main.java } ; if ($?) { java Main }
Current Balance: $500.0
Deposited: $200.0
Balance after deposit: $700.0
Withdrawn: $100.0
Balance after withdrawal: $600.0
Trying to withdraw $700.00
Withdrawal failed: Insufficient funds. Available balance: $600.0
Final Balance: $600.0
PS C:\Users\Shrey\OneDrive\Desktop\KJSCE\SEM-3\OOPS\Programs>
```



```
PS C:\Users\Shrey\OneDrive\Desktop\KJSCE\SEM-3\OOPS\Programs> cd "c:\Users\Shrey\OneDrive\Desktop\KJSCE\SEM-3\OOPS\Program
s\" ; if ($?) { javac Main.java } ; if ($?) { java Main }
Current Balance: $900.0
Deposited: $200.0
Balance after deposit: $1100.0
Withdrawn: $100.0
Balance after withdrawal: $1000.0
Trying to withdraw $1500.00
Withdrawal failed: Insufficient funds. Available balance: $1000.0
Final Balance: $1000.0
PS C:\Users\Shrey\OneDrive\Desktop\KJSCE\SEM-3\OOPS\Programs>
```