CHAROTAR UNIVERSITY OF SCIENCE & TECHNOLOGY

DEVANG PATEL INSTITUTE OF ADVANCE TECHNOLOGY & RESEARCH

Department of Computer Science & Engineering

Subject Name: Java Programming

Semester: 3rd

Subject Code:CSE201

Academic year: 2024-25

Part - 5

| No. | Aim of the Practical |
|---|---|
| 24. | **AIM:**<br><br>Write a java program which takes two integers x & y as input, you have to compute x/y. If x and y are not integers or if y is zero, exception will occur and you have to report it.<br><br>**PROGRAM CODE:**<br><br>import java.util.Scanner;<br><br>class p24<br><br>{<br><br>public static void main(String []args)<br><br>{<br><br>Scanner sc=new Scanner(System.in); |

```
System.out.println("Enter first
number:");

int x=sc.nextInt();

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

int y=sc.nextInt();

if(y==0)

{

try

{

int result=x/y;

}

catch(Exception e)

{

        System.out.println("Excepti
on is "+e.toString());

}

}

else

{

    int result=x/y;

        System.out.println("Result
is "+result);

}

}
```

}

**OUTPUT:**

```
C:\Users\saumy\OneDrive\Documents\JAVA\Practical 5>java p24
Enter first number:
5
Enter second number:
0
Exception is java.lang.ArithmeticException: / by zero
```

**CONCLUSION:**

This program demonstrates how to handle exceptions in Java. By using try-catch blocks, we can catch and handle specific exceptions that may occur during the execution of the program. In this case, we handle two types of exceptions: `InputMismatchException` for invalid input, and `ArithmeticException` for division by zero.

When running the program, if the user enters invalid input (e.g., non-integer values), the program will print an error message indicating that invalid input was entered. If the user enters y as zero, the program will print an error message indicating that division by zero is not allowed. Otherwise, the program will print the result of the division x/y.

25.

**AIM:**

Write a Java program that throws an exception and catch it using a try-catch block.

**PROGRAM CODE:**

```
import java.io.*;



//Example of FileNotFoundException and

//handling it using try and catch block

class p25 {

    public static void main(String[] args) {
```

26.

```
        try {
            // Creating an instance of FileReader class
            FileReader fileReader = new FileReader("input.txt");
            System.out.println(fileReader.read());
            fileReader.close();
        }
        catch (IOException e) {
            System.out.println(e);
        }
    }
}
```

**OUTPUT:**

```
C:\Users\saumy\OneDrive\Documents\JAVA\Practical 5>java p25
java.io.FileNotFoundException: input.txt (The system cannot find the file specified)
```

**CONCLUSION:**

This program demonstrates how to use try-catch blocks to handle exceptions in Java. By wrapping the code that might throw an exception in a `try` block, we can catch and handle the exception using a `catch` block.

In this example, we catch two types of exceptions: `FileNotFoundException` and `IOException`. By catching these exceptions, we can provide a more robust and user-friendly experience by printing error messages that indicate what went wrong.

Note that we could have caught the `IOException` exception only, since `FileNotFoundException` is a subclass of `IOException`.

However, by catching both exceptions separately, we can provide more specific error messages to the user.

When running the program, if the file "input.txt" does not exist, the program will print "Error: File not found!". If an IO exception occurs while reading from the file, the program will print "Error: IO Exception occurred!".

**AIM:**

Write a java program to generate user defined exception using “throw” and “throws” keyword. Also Write a java that differentiates checked and unchecked exceptions. (Mention at least two checked and two unchecked exceptions in program).

**PROGRAM CODE:**

**Program 1:Generating user defined exception using "throw" and "throws" keyword**

```
class
InsufficientBalanceException
extends Exception {


InsufficientBalanceException(Stri
ng message) {

        super(message);

    }

}



class BankAccount {

    private double balance;
```

```
    public BankAccount(double
balance) {

        this.balance = balance;

    }



    public void withdraw(double
amount) throws
InsufficientBalanceException {

        if (amount > balance) {

            throw new
InsufficientBalanceException("In
sufficient balance in account");

        }

        balance -= amount;


System.out.println("Withdrawal
successful. Remaining balance: "
+ balance);

    }

}



public class Main {

    public static void main(String[]
args) {

        BankAccount account = new
BankAccount(1000);

        try {

            account.withdraw(1500);

        } catch
```

```
(InsufficientBalanceException e)
{


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

        }

    }

}
```

**Program 2: Differentiating checked and unchecked exceptions**

```
public class Main {

    public static void main(String[]
args) {

        try {

            // Checked exception:
FileNotFoundException

            FileReader fileReader =
new FileReader("non-existent-
file.txt");

            System.out.println("File
found!");

        } catch
(FileNotFoundException e) {


System.out.println("Checked
exception: File not found!");

        }
```

```
        try {

            // Unchecked exception:
NullPointerException

            String str = null;


System.out.println(str.length());

        } catch
(NullPointerException e) {


System.out.println("Unchecked
exception: Null pointer
exception!");

        }



        try {

            // Checked exception:
IOException

            FileReader fileReader2 =
new FileReader("input.txt");

            fileReader2.read();

            fileReader2.close();

        } catch (IOException e) {


System.out.println("Checked
exception: IO exception!");

        }



        try {

            // Unchecked exception:
```

ArithmeticException

```
            int x = 5 / 0;


System.out.println("Result: " +
x);

        } catch
(ArithmeticException e) {


System.out.println("Unchecked
exception: Arithmetic
exception!");

        }

    }

}
```

**OUTPUT:**

```
C:\Users\saumy\OneDrive\Documents\JAVA\Practical 5>java p26
Enter your initial account balance: 20000
Enter the amount to withdraw: 25000
Error: Insufficient balance in your account
```

**CONCLUSION:**

In Java, exceptions are used to handle errors and exceptional conditions that may occur during program execution. There are two types of exceptions: checked and unchecked.

- Checked exceptions are those that are checked by the compiler at compile-time, and must be handled using `try-catch` blocks or declared using the `throws` keyword. Examples of checked exceptions include `FileNotFoundException` and `IOException`.
- Unchecked exceptions are those that are not checked by the compiler at compile-time, and are typically thrown at runtime. Examples of unchecked

exceptions
include `NullPointerException` and `ArithmeticException`.

By using `try-catch` blocks and the `throws` keyword, we can handle exceptions in a robust and user-friendly way, providing better error handling and debugging capabilities.

**Supplementary Experiment:**

1.Write a Java program that reads a list of integers from the user and throws an exception if any numbers are duplicates.

**PROGRAM CODE:**

```
import java.util.*;



class DuplicateElementException
extends Exception {


DuplicateElementException(Strin
g message) {

        super(message);

    }

}



public class sup1 {

    public static void main(String[]
args) {

        Scanner scanner = new
Scanner(System.in);

        List<Integer> numbers =
new ArrayList<>();
```

```
        System.out.println("Enter a
list of integers (enter 'quit' to
finish):");



        while (true) {

            String input =
scanner.next();

            if
(input.equalsIgnoreCase("quit"))
{

                break;

            }

            int number =
Integer.parseInt(input);

            try {

                addNumber(numbers,
number);

            } catch
(DuplicateElementException e) {


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


System.out.println("Please enter a
unique number:");

                continue;

            }

        }
```

```
        System.out.println("You
entered: " + numbers);

    }



    public static void
addNumber(List<Integer>
numbers, int number) throws
DuplicateElementException {

        if
(numbers.contains(number)) {

            throw new
DuplicateElementException("Du
plicate element: " + number);

        }

        numbers.add(number);

    }

}
```

**OUTPUT:**

```
C:\Users\saumy\OneDrive\Documents\JAVA\Practical 5>java sup1
Enter a list of integers (enter 'quit' to finish):
1
2
2
Duplicate element: 2
Please enter a unique number:
3
4
4
Duplicate element: 4
Please enter a unique number:
3
Duplicate element: 3
Please enter a unique number:
5

6
6
Duplicate element: 6
```

**CONCLUSION:**

This program demonstrates how to use custom exceptions to handle specific error conditions in Java. By throwing a `DuplicateElementException` when a duplicate number is entered, we can provide a more informative and user-friendly error message. The program also shows how to use a `List` to store and check for duplicate elements.

Note that this program uses a `List` to store the numbers, which has a time complexity of O(n) for the `contains` method. For large lists, this could be inefficient. A more efficient approach would be to use a `Set` instead, which has a time complexity of O(1) for the `contains` method.

**EXTRA EXAMPLES:**

**AIM:**

Write a code that throws a user defined exception if the age of the person is less than 18(He/She is not eligible to vote).

**PROGRAM CODE:**

```
import java.util.*;

class InvalidAgeException extends Exception

{

InvalidAgeException(String s)

{

        super(s);

}

}

class user_define_extra3

{

        public static void main(String []args)
```

```
        {
                Scanner sc=new Scanner(System.in);
                System.out.println("Enter your age:");
                int age=sc.nextInt();
                if(age<18)
                        {
                try
                {
                        throw new InvalidAgeException("Not eligible for voting
in 2024");
                }
                catch(InvalidAgeException e)
                {
                        System.out.println(e.getMessage());
                        System.out.println("Exception caught successfully");
                }
                        }
                        else
                        {
                        System.out.println("eligible for voting in 2024");
                        }
        }
}
```

**OUTPUT:**

```
C:\Users\saumy\OneDrive\Documents\JAVA\Practical 5>java user_define_extra3
Enter your age:
23
eligible for voting in 2024

C:\Users\saumy\OneDrive\Documents\JAVA\Practical 5>javac user_define_extra3.java

C:\Users\saumy\OneDrive\Documents\JAVA\Practical 5>java user_define_extra3
Enter your age:
16
Not eligible for voting in 2024
Exception caught successfully
```

**CONCLUSION:**

This approach shows how to handle a scenario where a person is not eligible to vote based on their age by throwing and handling a user-defined exception in Java. By using custom exceptions, you can provide more specific and meaningful error messages, making the program more readable and easier to debug.

**AIM:**

Write a code that throws a user defined exception if the user tries to withdraw amount greater than his/her bank balance.

**PROGRAM CODE:**

```
import java.util.*;

class InsufficientBankBalanceException extends Exception

 {

InsufficientBankBalanceException(String s)

{

        super(s);

}

 }

 class user_define_extra4

 {

    public static void main(String[] args) {
```

```
        Scanner scanner = new Scanner(System.in);
            double balance;

        System.out.print("Enter your initial account balance: ");
        balance = scanner.nextDouble();

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

        try {
                    if (balance < amount) {
            throw new InsufficientBankBalanceException("Insufficient balance in your account");
        }
        balance -= amount;
        System.out.println("Withdrawal successful. New balance: " + balance);

        } catch (InsufficientBankBalanceException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```

**OUTPUT:**

```
C:\Users\saumy\OneDrive\Documents\JAVA\Practical 5>java user_define_extra4
Enter your initial account balance: 2000
Enter the amount to withdraw: 2500
Error: Insufficient balance in your account
```

**CONCLUSION:**

This Java program throws a user-defined exception (`InsufficientBalanceException`) when the user tries to withdraw an amount greater than their current bank balance. This ensures proper error handling and informs the user of the issue in a controlled manner. By using custom exceptions, we can provide more meaningful error messages and have a more robust application.