# CHAROTAR UNIVERSITY OF SCIENCE & TECHNOLOGY

## DEVANG PATEL INSTITUTE OF ADVANCE TECHNOLOGY & RESEARCH

Department of Computer Science Engineering

**Subject Name: Java programming**
**Semester: III**
**Subject Code: CSE201**
**Academic year: 2024-25**

# Part - 5

| No. | Aim of the Practical |
|---|---|
| **24.** | 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>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);<br><br>System.out.println("Enter first number:");<br><br>int x=sc.nextInt();<br><br>System.out.println("Enter second number:");<br><br>int y=sc.nextInt(); |

```
if(y==0)
{
try
{
int result=x/y;
}
catch(Exception e)
{
        System.out.println("Exception is
"+e.toString());
}
}
else
{
   int result=x/y;
        System.out.println("Result is
"+result);
}
}
}
```

**OUTPUT:**

```
D:\java>javac pra24.java

D:\java>java pra24
Enter the first integer (x): 3
Enter the second integer (y): 4
x / y = 0

D:\java>javac pra24.java

D:\java>java pra24
Enter the first integer (x): 3
Enter the second integer (y): 0
Exception solved 1java.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.

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

**PROGRAM CODE:**

```
public class pra25 {
   public static void main(String[] args) {
      try {
         int[] a = {1, 2, 3};
         System.out.println(a[10]);
      } catch (Exception e) {
         System.out.println("Something went wrong: " + e.getMessage());
      }
   }
}
```

**OUTPUT:**

```
D:\java>javac pra25.java

D:\java>java pra25
Something went wrong: Index 10 out of bounds for length 3

D:\java>
```

**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.

**26.**

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:**

```
import java.util.Scanner;

class InsufficientBalanceException extends Exception {
   InsufficientBalanceException(String message) {
      super(message);
   }
}

public class insufficientbank throws InsufficientBalanceException {
   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 InsufficientBalanceException("Insufficient balance in your
account");
      }
      balance -= amount;
      System.out.println("Withdrawal successful. New balance: " + balance);

      } catch (Exception e) {
```

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

**OUTPUT:**

```
D:\java>javac insufficientbank.java

D:\java>java insufficientbank
Enter your initial account balance: 20000
Enter the amount to withdraw: 5000
Withdrawal successful. New balance: 15000.0

D:\java>javac insufficientbank.java

D:\java>java insufficientbank
Enter your initial account balance: 5000
Enter the amount to withdraw: 20000
Error: Insufficient balance in your account

D:\java>
```

**CONCLUSION:**

The modified code uses the throws keyword to indicate that the main method may throw an InsufficientBalanceException if the account balance is insufficient for withdrawal.

**Sup 26.**

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

**PROGRAM CODE:**

```
import java.util.Scanner;

class DuplicateNumberException extends Exception {
    DuplicateNumberException(String message) {
        super(message);
    }
}

public class sup26 {
    public static void checkForDuplicates(int[] numbers) throws
DuplicateNumberException {
        for (int i = 0; i < numbers.length; i++) {
            for (int j = i + 1; j < numbers.length; j++) {
                if (numbers[i] == numbers[j]) {
```

```
                throw new DuplicateNumberException("Duplicate number found: " +
numbers[i]);
            }
        }
    }
}

public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);

    System.out.print("Enter the number of integers: ");
    int count = scanner.nextInt();

    int[] numbers = new int[count];

    System.out.println("Enter the integers:");
    for (int i = 0; i < count; i++) {
        numbers[i] = scanner.nextInt();
    }

    try {
        checkForDuplicates(numbers);
        System.out.println("All numbers are unique.");
    } catch (DuplicateNumberException e) {
        System.out.println(e.getMessage());
    }
}
}
```

**OUTPUT:**

```
D:\java>javac sup26.java

D:\java>java sup26
Enter the number of integers: 5
Enter the integers:
1
4
2
3
3
Duplicate number found: 3
```

**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.

**extra**

Insufficient bank balance using throws

**PROGRAM CODE**

```
import java.util.Scanner;

class InsufficientBalanceException extends Exception {
   InsufficientBalanceException(String message) {
      super(message);
   }
}

public class insufficientbank throws InsufficientBalanceException {
   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 InsufficientBalanceException("Insufficient balance in your
account");
      }
      balance -= amount;
      System.out.println("Withdrawal successful. New balance: " + balance);

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

**OUTPUT**:

```
D:\java>javac insufficientbank.java

D:\java>java insufficientbank
Enter your initial account balance: 5000
Enter the amount to withdraw: 15000
Error: Insufficient balance in your account

D:\java>
```

**CONCLUSION**

The code provides a basic implementation of a banking system that handles withdrawals and checks for insufficient balance. However, there are some areas for improvement:

- The code does not handle cases where the user enters invalid input (e.g., non-numeric values).
- The code does not provide a way to deposit money into the account.
- The code does not store the account balance persistently, so it will be lost when the program terminates.

**extra**

To check if the person is eligible to vote or not.

**PROGRAM CODE**

```
import java.util.Scanner;

class InvalidAgeException extends Exception {
    InvalidAgeException(String message) {
        super(message);
    }
}

public class Voter {
    int age;

    public static void main(String[] args) {
        Voter voter = new Voter();
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter your age: ");
        voter.age = scanner.nextInt();

        try {
```

```
            if (voter.age < 18) {
                throw new InvalidAgeException("You are not eligible for voting in
2024");
            }
            System.out.println("You have successfully cast your vote!");
        }
                catch (InvalidAgeException e)
                {
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```

**OUTPUT**

```
D:\java>javac Voter.java

D:\java>java Voter
Enter your age: 12
Error: You are not eligible for voting in 2024
```

**CONCLUSION**

The provided Java code is a simple implementation of a voting system that checks if a person is eligible to vote based on their age. Here's a breakdown of the code:

- A custom exception class **InvalidAgeException** is created to handle cases where the user's age is less than 18.
- In the **main** method, a **Voter** object is created, and a **Scanner** object is used to get the user's age from the console.
- The code checks if the user's age is less than 18. If true, it throws an **InvalidAgeException** with a custom error message.
- If the age is 18 or above, it prints a success message indicating that the user has successfully cast their vote.
- The **try-catch** block is used to handle the **InvalidAgeException** and print the error message if the exception is thrown.