Exercise 7

Hariesh R - 23110344

Aim:

1. The aim of this Java program is to check whether the provided array size is negative and handle it using try and catch blocks. If a negative size is entered, a NegativeArraySizeException is thrown, which is caught by the catch block, displaying an appropriate error message. This prevents the program from crashing due to invalid input, ensuring graceful error handling when a negative array size is provided.

2. The aim of this Java program is to create a method that checks if the input age is valid for voting, where the minimum required age is 18. If the input is less than 18, the method throws an exception using the throws keyword. The exception is then handled appropriately, ensuring that invalid ages (less than 18) are caught and processed without causing the program to crash. This approach enforces proper age validation using exception handling mechanisms in Java.

3. The aim of this Java program is to demonstrate the use of multiple catch statements to handle different types of exceptions that may occur within a try block. The program handles various exceptions, such as ArithmeticException (e.g., division by zero) and ArrayIndexOutOfBoundsException (e.g., accessing an invalid array index). Each specific exception is caught by its corresponding catch block, ensuring that the program can manage multiple types of errors gracefully without crashing.

4. The aim of this Java program is to demonstrate exception handling using the throw keyword for explicitly throwing exceptions, and the finally block to ensure code execution regardless of exceptions. This ensures proper error handling and guarantees that necessary cleanup or final actions are always executed.

Algorithm:

1) Q1:
    i) Start the program.

ii) Initialize scanner:
    (a) Create a Scanner object to read user input.
iii) Prompt for array size:
    (a) Display a message "Enter size:" to prompt the user for input.
    (b) Read the integer input for the array size.
iv) Try block:
    (a) In the try block, attempt to create an array of the specified size.
    (b) If the array is successfully created, print a success message along with the array size.
v) Catch block (exception handling):
    (a) If a NegativeArraySizeException occurs (i.e., if the input size is negative), catch the exception.
    (b) Print an error message containing the exception details.
vi) Close scanner:
    (a) Close the Scanner object to prevent resource leaks.
vii) End the program.

2) Q2:
   i) Start the program.
   ii) Define CustomError class:
       (a) Create a custom exception class CustomError that extends Exception and takes a message as input.
   iii) Initialize scanner:
       (a) Create a Scanner object to read user input.
   iv) Prompt for age:
       (a) Display a message "Enter the age:" to prompt the user for input.
       (b) Read the integer input for the age.
   v) Try block:
       (a) In the try block, call the checkVotingAge method, passing the user's input (age) as a parameter.
   vi) Catch block (CustomError handling):
       (a) If a CustomError is thrown (age is less than 18), catch the exception and print the error message.
   vii) Method checkVotingAge:
       (a) If the input age is less than 18, throw a CustomError with the message "Not Eligible To Vote."
       (b) Otherwise, print "Eligible To Vote."
   viii) Close scanner:
       (a) Close the Scanner object to prevent resource leaks.
   ix) End the program.

3) Q3:
   i) Start the program.

ii) Initialize an array of strings:
    (a) Create an array inputs containing string values: "100", "abc", "12.34", and null.

iii) Iterate through the array:
    (a) For each input string in the array, proceed with the following steps.

iv) Try block:
    (a) Attempt to convert the input string to an integer using Integer.parseInt(input).
    (b) Attempt to divide the converted integer by 0.
    (c) If no exception occurs, print the result of the division.

v) Catch block (multiple exceptions):
    (a) NumberFormatException: If the input is not a valid integer (e.g., "abc", "12.34"), catch the exception and print an error message about invalid number format.
    (b) ArithmeticException: If a division by zero occurs, catch the exception and print an error message about the arithmetic exception.
    (c) NullPointerException: If the input is null, catch the exception and print an error message about null pointer access.
    (d) Exception: Catch any other unexpected exceptions and print a generic error message.

vi) End the loop and the program.

4) Q4:

i) Start the program.

ii) Initialize scanner:
    (a) Create a Scanner object to read user input.

iii) Prompt for numerator:
    (a) Display a message "Enter numerator:" and read the integer input for the numerator.

iv) Prompt for divisor:
    (a) Display a message "Enter divisor:" and read the integer input for the divisor.

v) Close scanner:
    (a) Close the Scanner object to prevent resource leaks.

vi) Try block:
    (a) Call the performDivision method, passing the numerator and divisor as parameters.
    (b) Catch block (ArithmeticException handling):
    (c) If an ArithmeticException occurs (e.g., division by zero), catch the exception and print an error message.

vii) Finally block:
    (a) Execute the finally block, printing a message indicating it has been executed, regardless of whether an exception was caught.

viii) Method performDivision:
    (a) If the divisor is 0, throw an ArithmeticException with the message "Cannot divide by zero."
    (b) Otherwise, perform the division of the numerator by the divisor and print the result.

ix) End the program.

Source Code:

1) Q1:

```
package Exercise7;

import java.util.Scanner;

public class Q1 {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter size: ");
        int size = scanner.nextInt();
        scanner.nextLine();

        try {

            int[] Array = new int[size];
            System.out.println("Successfully Created !\nSize: " + Array.length);
        }

        catch(NegativeArraySizeException exception){

            System.out.println("Error !\nReceived: " + exception.getMessage());
        }

        scanner.close();
    }
}
```

2) Q2:

```
package Exercise7;
import java.util.Scanner;

class CustomError extends Exception{
    public CustomError(String message){

        super(message);
    }
}

public class Q2 {


    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the age: ");
        int age = scanner.nextInt();
        scanner.nextLine();

        try{

            checkVotingAge(age);
        }

        catch(CustomError error){

            System.out.println(error.getMessage());
        }

        scanner.close();
    }

    public static void checkVotingAge(int age) throws CustomError{

        if (age < 18){

            throw new CustomError("Not Eligible To Vote");
        }

        else
            System.out.println("Eligible To Vote");
    }
}
```

3) Q3:

```
package Exercise7;
```

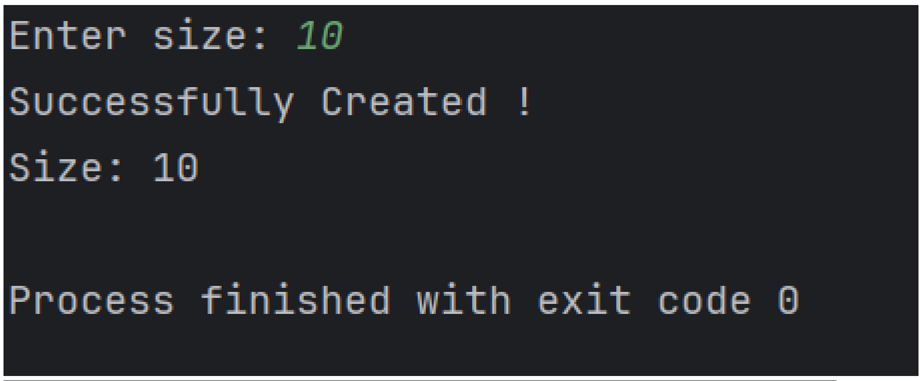
```
public class Q3 {

    public static void main(String[] args) {

        String[] inputs = {"100", "abc", "12.34", null};

        for (String input : inputs) {

            try {

                int number = Integer.parseInt(input);

                int result = number / 0;

                System.out.println("Result of division: " + result);

            } catch (NumberFormatException e) {

                System.out.println("Error: Invalid number format. " + e.getMessage());

            } catch (ArithmeticException e) {

                System.out.println("Error: Arithmetic exception. " + e.getMessage());

            } catch (NullPointerException e) {

                System.out.println("Error: Null pointer exception. " + e.getMessage());

            } catch (Exception e) {

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

4) Q4:

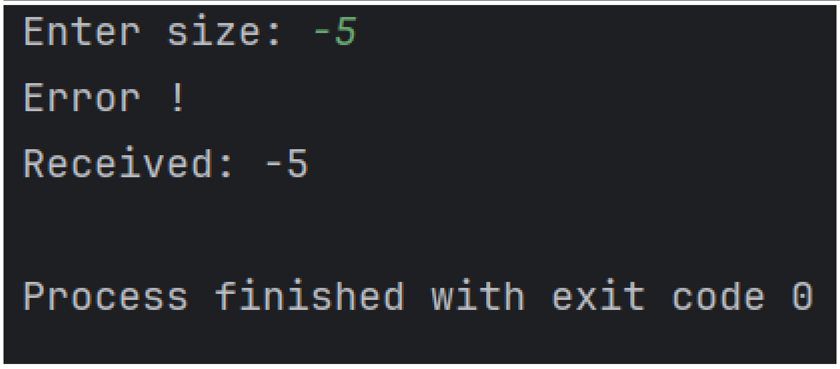
```
package Exercise7;

import java.util.Scanner;

public class Q4{

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter numerator: ");
        int numerator = scanner.nextInt();
        scanner.nextLine();

        System.out.print("Enter divisor: ");
        int divisor = scanner.nextInt();
        scanner.nextLine();

        scanner.close();

        try {

            performDivision(numerator, divisor);
        } catch (ArithmeticException e) {

            System.out.println("Caught an exception: " + e.getMessage());
        } finally {

            System.out.println("Finally block executed.");
        }
    }

    public static void performDivision(int numerator, int divisor) throws
ArithmeticException{

        if (divisor == 0) {

            throw new ArithmeticException("Cannot divide by zero");
        }

        int result = numerator / divisor;
        System.out.println("Result of division: " + result);
    }
}
```

Output:

1) Q1:

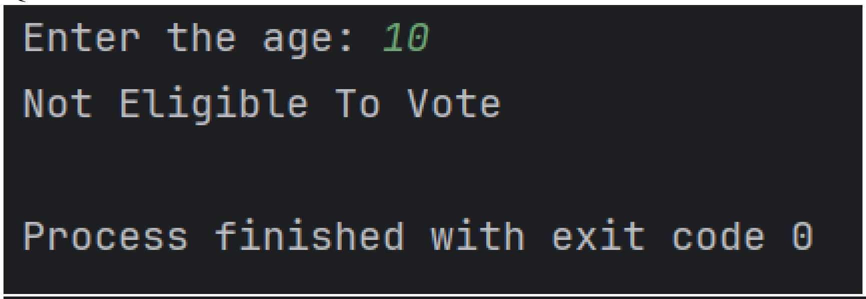


```
Enter size: 10
Successfully Created !
Size: 10

Process finished with exit code 0
```

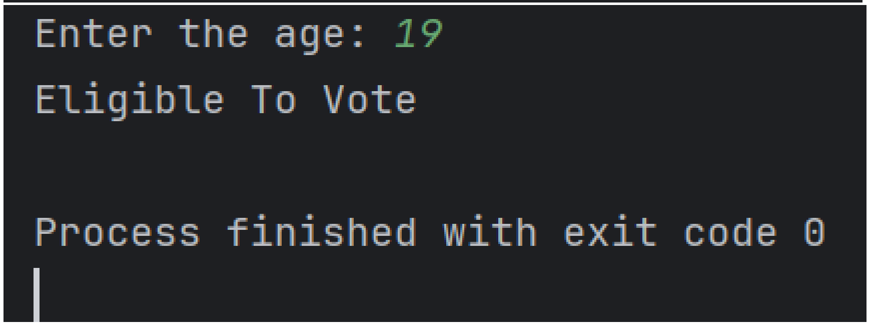


```
Enter size: -5
Error !
Received: -5

Process finished with exit code 0
```

2) Q2:

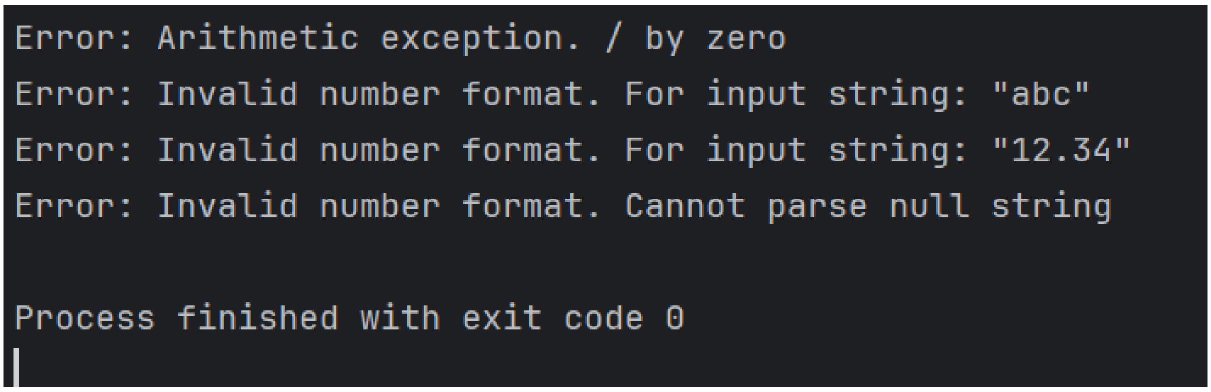


```
Enter the age: 10
Not Eligible To Vote

Process finished with exit code 0
```



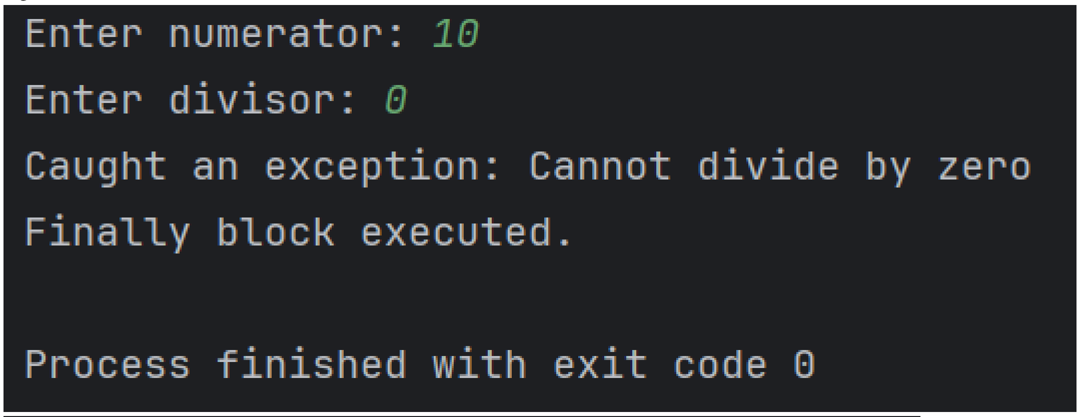
```
Enter the age: 19
Eligible To Vote

Process finished with exit code 0
```

3) Q3:

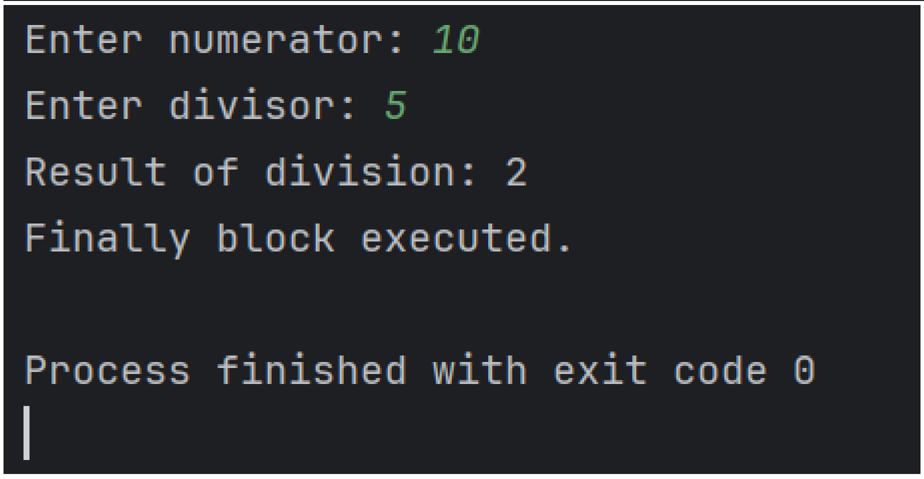


```
Error: Arithmetic exception. / by zero
Error: Invalid number format. For input string: "abc"
Error: Invalid number format. For input string: "12.34"
Error: Invalid number format. Cannot parse null string

Process finished with exit code 0
```

4) Q4:



```
Enter numerator: 10
Enter divisor: 0
Caught an exception: Cannot divide by zero
Finally block executed.

Process finished with exit code 0
```



```
Enter numerator: 10
Enter divisor: 5
Result of division: 2
Finally block executed.

Process finished with exit code 0
```

Result:

1. The result of the Java program that checks whether the given array size is negative involves handling an exception if a negative size is entered. If the user inputs a non-negative size, the program successfully creates the array and displays a success message with the array size. However, if the user provides a negative size, the program catches the NegativeArraySizeException and

displays an error message indicating that the array size cannot be negative. This ensures that the program handles invalid input gracefully without crashing.

2. The result of the Java program that checks voting eligibility involves using a method to determine if the input age is valid for voting. If the age is less than 18, the method throws a CustomError exception, which is then caught and handled in the catch block of the main program. If the age is 18 or older, the program prints a message indicating eligibility to vote. The use of the throws keyword ensures that the CustomError exception is properly handled, while providing clear feedback on age validation.

3. The result of the Java program demonstrating multiple catch statements is that it handles various exceptions that may arise from a single try block. For each input processed, the program attempts to parse it as an integer and perform a division operation. If the input causes a NumberFormatException, ArithmeticException, NullPointerException, or any other unexpected exception, the corresponding catch block handles it by printing a specific error message. This ensures that different types of errors are managed appropriately, providing clear feedback for each type of exception encountered.

4. The result of the Java program demonstrating exception handling with the throw keyword and finally block involves explicitly throwing an exception when a certain condition is met and ensuring that a finally block always executes. In the program, an exception (e.g., IllegalArgumentException) is thrown if a specific condition (e.g., invalid input) occurs. Regardless of whether an exception is thrown or not, the finally block runs to perform cleanup or final actions, such as closing resources or printing a message. This approach guarantees that essential code is executed even if an error occurs.