Kartik Verma

CSE-2C

Java Lab File

Lab 5:-

Program 13: **Handling ArithmeticException-** Write a java program that takes two integers as input from the user and divides the first number by the second. Use exception handling to manage the ArithmeticException that occurs when the second number is zero.

Software Used:- VS Code

Code:-

```
import java.util.Scanner;
public class DivisionWithExceptionHandling {
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in){
      try {
         System.out.print("Enter the first number (numerator): ");
         int numerator = scanner.nextInt();

         System.out.print("Enter the second number (denominator): ");
         int denominator = scanner.nextInt();

         int result = numerator / denominator;

         System.out.println("The result of the division is: " + result);
      } catch (ArithmeticException e) {
         // Handling the ArithmeticException when the denominator is zero
         System.out.println("Error: Division by zero is not allowed.");
```

```
        } finally {
            // Closing the scanner
            scanner.close();
        }
    }
}
```

Output:-

```
Enter the first number (numerator): 10
Enter the second number (denominator): 2
The result of the division is: 5

Enter the first number (numerator): 10
Enter the second number (denominator): 0
Error: Division by zero is not allowed.

Enter the first number (numerator): 25
Enter the second number (denominator): 5
The result of the division is: 5

Enter the first number (numerator): -15
Enter the second number (denominator): 3
The result of the division is: -5
```

Kartik Verma

CSE-2C

Java Lab File

Lab 5:-

Program 14: **Custom Exception Class**- Create a custom exception class named NegativeNumberException that extends Exception. Write a program that takes an integer as input and throws NegativeNumberException if the number is negative. Handle this exception and display an appropriate message.

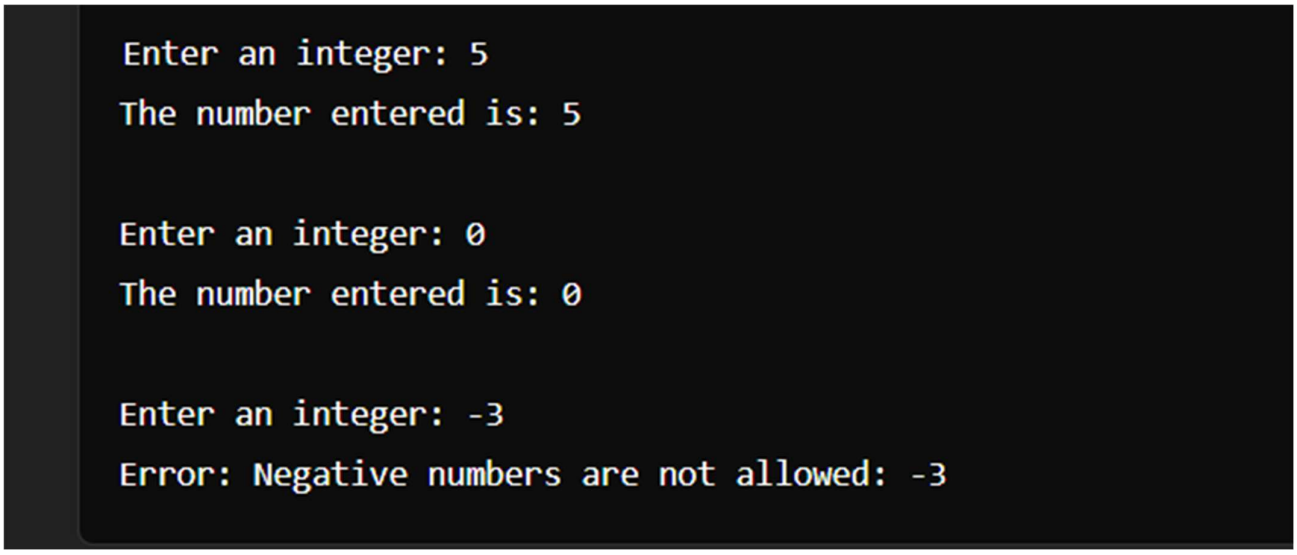
Software Used:- VS Code

Code:-

```
import java.util.Scanner;

public class NegativeNumberTest {

   public static void main(String[] args) {

      Scanner scanner = new Scanner(System.in);

         try {

         // Taking an integer as input from the user

         System.out.print("Enter an integer: ");

         int number = scanner.nextInt();


         // Checking if the number is negative

         if (number < 0) {

            throw new NegativeNumberException("Negative numbers are not
allowed: " + number);

         }


         // If the number is not negative, print the number

         System.out.println("The number entered is: " + number);
```

```
        } catch (NegativeNumberException e) {
            // Handling the custom exception and displaying an appropriate
message
            System.out.println("Error: " + e.getMessage());
        } finally {
            // Closing the scanner
            scanner.close();
        }
    }
}
```

Output:-



Kartik Verma

CSE-2C

Java Lab File

Lab 5:-

Program 15: **Multiple Catch Blocks**- Write a program that takes an array of integers and an index from the user. Display the element at the specified index. Use multiple catch blocks to handle ArrayIndexOutOfBoundsException and NumberFormatException.

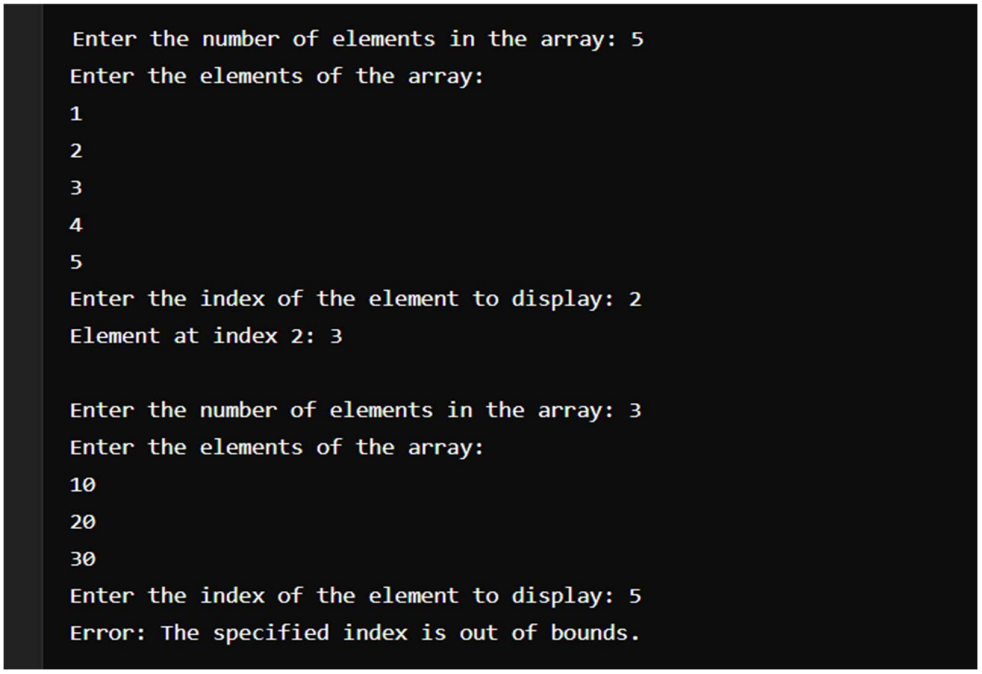
Software Used:- VS Code

Code:-

```
import java.util.Scanner;

public class ArrayElementFetcher {

   public static void main(String[] args) {

      Scanner scanner = new Scanner(System.in);


      try {

         // Taking the number of elements in the array from the user

         System.out.print("Enter the number of elements in the array: ");

         int arraySize = scanner.nextInt();

         int[] array = new int[arraySize];


         // Taking the elements of the array from the user

         System.out.println("Enter the elements of the array:");

         for (int i = 0; i < arraySize; i++) {

            array[i] = scanner.nextInt();

         }


         // Taking the index from the user
```

```
            System.out.print("Enter the index of the element to display: ");
            int index = scanner.nextInt();

            // Displaying the element at the specified index
            System.out.println("Element at index " + index + ": " + array[index]);
        } catch (ArrayIndexOutOfBoundsException e) {
            // Handling ArrayIndexOutOfBoundsException
            System.out.println("Error: The specified index is out of bounds.");
        } catch (NumberFormatException e) {
            // Handling NumberFormatException
            System.out.println("Error: Invalid input. Please enter valid integers.");
        } catch (Exception e) {
            // Handling any other exceptions
            System.out.println("Error: An unexpected error occurred.");
        } finally {
            // Closing the scanner
            scanner.close();
        }
    }
}
```

Output:-



Kartik Verma

CSE-2C

Java Lab File

Lab 5:-

Program 16: **Creating a Thread by Extending Thread Class**- Create a class MyThread that extends Thread and overrides the run() method to print numbers from 1 to 10. In the main method, create and start an instance of MyThread.

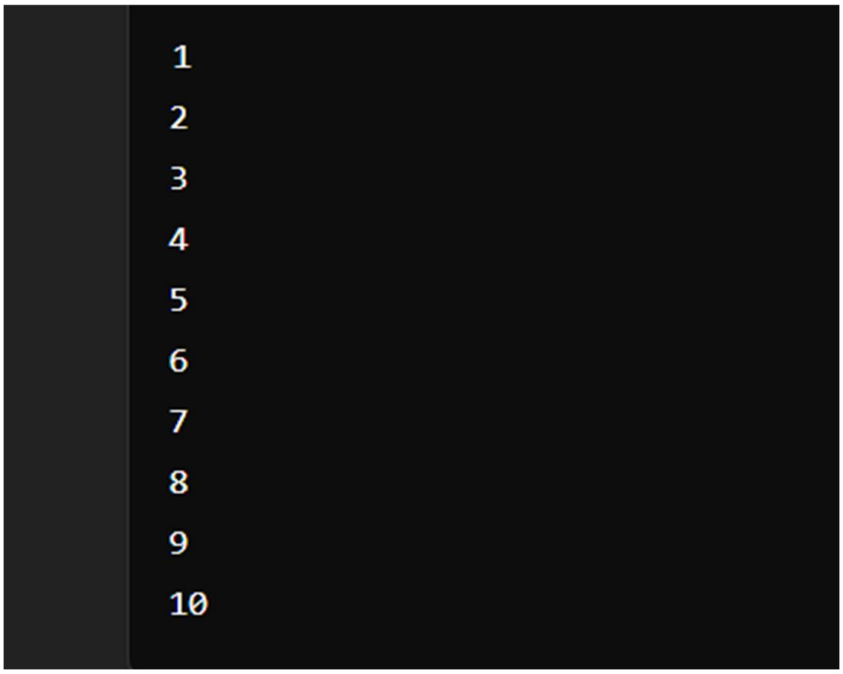
Software Used:- VS Code

Code:-

```
class MyThread extends Thread {
   @Override
   public void run() {
      // Printing numbers from 1 to 10
      for (int i = 1; i <= 10; i++) {
         System.out.println(i);
      }
   }

       public static void main(String[] args) {
      // Creating an instance of MyThread
      MyThread myThread = new MyThread();

      // Starting the thread
```

```
        myThread.start();
    }
}
```

Output:-



Kartik Verma

CSE-2C

Java Lab File

Lab 5:-

Program 17: **Implementing Runnable Interface-** Create a class MyRunnable that implements Runnable and overrides the run() method to print "Thread is running". In the main method, create an instance of Thread passing a MyRunnable object and start the thread.

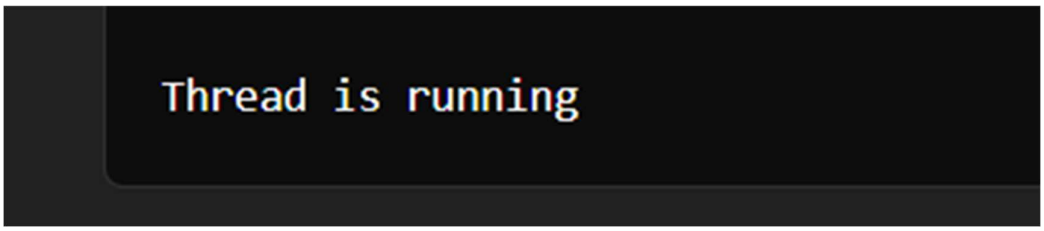
Software Used:- VS Code

Code:-

```
class MyRunnable implements Runnable {
    @Override
    public void run() {
        // Printing a message to indicate the thread is running
        System.out.println("Thread is running");
    }

    public static void main(String[] args) {
        // Creating an instance of MyRunnable
        MyRunnable myRunnable = new MyRunnable();


        // Creating an instance of Thread, passing MyRunnable instance
        Thread thread = new Thread(myRunnable);
```

```
        // Starting the thread
        thread.start();
    }
}
```

Output:-



Kartik Verma

CSE-2C

Java Lab File

Lab 5:-

Program 18: **Synchronized Method**- Create a class Counter with a method increment() that increments a variable count by 1 and prints the current value of count. Use synchronization to ensure that multiple threads can safely increment the counter without data inconsistency. Create and start two threads that call the increment() method.

Software Used:- VS Code

Code:-

```
class Counter {
   private int count = 0;
 // Synchronized method to increment count
   public synchronized void increment() {
      count++;
      System.out.println("Current value of count: " + count);
   }
   public static void main(String[] args) {
      // Creating an instance of Counter
      Counter counter = new Counter();

      // Creating and starting two threads that call increment()
method
      Thread thread1 = new Thread(new Runnable() {
```

```
            public void run() {
                counter.increment();
            }
        });

        Thread thread2 = new Thread(new Runnable() {
            public void run() {
                counter.increment();
            }
        });

        // Starting the threads
        thread1.start();
        thread2.start();
    }
}
```

Output:-