UNIVERSIDAD DE LA FRONTERA
FACULTAD DE INGENIERÍA Y CIENCIAS

Actividad:
“Números Adyacentes”

PROFESOR:

Samuel Eduardo Sepulveda Cuevas

INTEGRANTES:

Arturo Avalos

Lian Canio

Sebastián Llanos

Introducción:

En el presente informe se documentará el proceso realizado para el desarrollo de la actividad asociada a “Números Adyacentes”, mejorando la aplicación de buenas prácticas de la programación, uso de pruebas unitarias, manejo de excepciones y el uso del cronómetro para tener una percepción del tiempo utilizado en cada actividad y la posible mejora de las habilidades de programación.

Descripción del caso

El problema planteado consiste en encontrar el mayor producto entre dos números enteros adyacentes contenidos en una lista o arreglo de más números. Esta lista de números debe ser de un largo igual o mayor que 2, y los números ingresados en ella deben estar entre -1.000 y 1.000.

Análisis del caso

El problema de encontrar el mayor producto de números adyacentes se puede solucionar recorriendo el arreglo de enteros con una estructura de control iterativo como “fori” y calculando el producto de cada par de números adyacentes. Esto mediante un método que reciba como parámetro de entrada un arreglo, y deberá retornar un entero que corresponda al mayor producto entre adyacentes. El programa deberá mantener un seguimiento del mayor producto encontrado hasta ese momento y actualizarlo cada vez que se encuentre un nuevo producto mayor.

Además, el programa debe tener en cuenta posibles casos especiales, como arreglos con valores negativos y valores nulos. Además, se deben implementar pruebas unitarias para garantizar la corrección del programa en diferentes escenarios.

Implementación de la solución:

Nuestro código consta de 4 métodos los cuales son: “main”, “entradaLargoArray”, “entradaArray” y “productoAdyacentes”. El método “main” principalmente llama a las otras funciones para poder imprimir el resultado de la multiplicación de los 2 números adyacentes. El método “entradaLargoArray” mediante un scanner, permite a un usuario ingresar el largo de su lista y válida si su respuesta es menor a 2, en ese caso le pide ingresarla nuevamente. El método “entradaArray” le pide al usuario mediante scanners que ingrese los números en orden que irán dentro de la lista. Por ultimo el método “productoAdyacentes” con un fori, multiplica la posición i de la lista con la posición i+1, en el caso de ser mayor que el resultado ya guardado, cambia esta nueva multiplicación a la variable resultado.

```
package org.example;
import java.util.Scanner;

public class casoAdyacentes {
    public static void main(String[] args) {
        int[] array = entradaArray(entradaLargoArray());
        System.out.println("El mayor resultado de la multiplicación entre 2 numeros adyacentes es de: ");
        System.out.print(productoAdyacentes(array));
    }


    public static int entradaLargoArray() {
        boolean esNumero;
        int largo=0;
        Scanner sc = new Scanner(System.in);
        do{
            try{
                esNumero=true;
                System.out.print("Ingrese el largo del array: ");
                largo = sc.nextInt();
            } catch (Exception e){
                sc.next();
                System.out.println("Tienen que ser numeros enteros, ingreselos nuevamente");
                esNumero=false;
            }
        } while(!esNumero);
        while (largo<2){
            System.out.println("No es válido un largo de "+largo+", ingrese un largo mayor que 1:");
            largo=sc.nextInt();
```

```
        }
        return largo;
    }
    public static int[] entradaArray(int largo) {
        boolean esNumero;
        Scanner sc = new Scanner(System.in);
        int[] arreglo = new int[largo];
        for (int i = 0; i < arreglo.length; i++) {
            do{
                try{
                    esNumero=true;
                    System.out.print("Ingrese el valor de la posición " + (i+1) + ": ");
                    arreglo[i] = sc.nextInt();
                } catch (Exception e){
                    sc.next();
                    System.out.println("Tienen que ser numeros enteros, ingreselos nuevamente");
                    esNumero=false;
                }
            } while(!esNumero);
        }
        return arreglo;
    }
    public static int productoAdyacentes(int[] arrayNum) {
        int producto = arrayNum[0] * arrayNum[1];
        for (int i = 0; i < arrayNum.length - 1; i++) {
            if (arrayNum[i] * arrayNum[i+1] > producto) {
                producto = arrayNum[i] * arrayNum[i+1];
            }
        }
        return producto;
    }
}
```

Diseño, implementación y resultados de las pruebas unitarias:

Se realizaron diversas pruebas para el códigos con diferentes arreglos establecidos y todas las pruebas unitarias realizadas dieron los resultados esperados, siendo un éxito en la implementación de nuestro código.

```
package org.example;
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;

class casoAdyacentesTest {

    @Test
    void productoAdyacentes() {
        int[] arreglo = {1, 0, 3, 7, 0, 5};
        int[] arreglo2 = {-3,2,-5,1,-8,2};
        int[] arreglo3 = {2,2,2,2,2};
        int[] arreglo4 = {7,3};
        assertEquals(21, casoAdyacentes.productoAdyacentes(arreglo));
        assertEquals(-5, casoAdyacentes.productoAdyacentes(arreglo2));
        assertEquals(4, casoAdyacentes.productoAdyacentes(arreglo3));
        assertEquals(21 , casoAdyacentes.productoAdyacentes(arreglo4));
    }
}
```

Diseño, implementación y resultados de las excepciones implementadas:

Creamos excepciones para 2 problemas, estos problemas ocurrían cuando el usuario no ingresaba un número entero, y cuando el largo era menor a 2. Para resolverlas realizamos un do while anexado de un try catch, el do while no te permita salir del bucle mientras la condición fuese falsa, y esta condición solo era falsa si el catch detectaba una exception, y para resolver el segundo problema realizamos un while que cuando el largo fuese menor a 2, entraba a un bucle y el usuario no podía salir del bucle hasta que definiese el largo como 2 o mayor que 2.

```
do{
```

```
            try{
                esNumero=true;
                System.out.print("Ingrese el largo del array: ");
                largo = sc.nextInt();
            } catch (Exception e){
                sc.next();
                System.out.println("Tienen que ser numeros enteros, ingreselos nuevamente");
                esNumero=false;
            }
        } while(!esNumero);
while (largo<2){
            System.out.println("No es valido un largo de "+largo+", ingrese un largo mayor que 1:");
            largo=sc.nextInt();
}



do{
                try{
                    esNumero=true;
                    System.out.print("Ingrese el valor de la posición " + (i+1) + ": ");
                    arreglo[i] = sc.nextInt();
                } catch (Exception e){
                    sc.next();
                    System.out.println("Tienen que ser numeros enteros, ingreselos nuevamente");
                    esNumero=false;
                }
            } while(!esNumero);
```

Discusión con los resultados y comentarios de la experiencia:

Ante las pruebas realizadas al programa, podemos decir que es funcional y devuelve una respuesta aceptable ante las excepciones planteadas. El código resultante no es de gran dificultad, y las experiencias obtenidas mediante la creación de este se puede destacar la mejora en el uso de arreglos, estructuras de iteración, manejo de excepciones y pruebas unitarias. Algo importante a destacar es el uso del cronómetro para registrar el tiempo real utilizado para la creación de la solución/código, ya que nos hace reflexionar sobre el porqué de la variación de este tiempo y el cómo mejorarlo.

Conclusiones:

Se logró abordar con éxito el problema planteado, estableciendo una solución completamente funcional que cumple con los requisitos solicitados. El desarrollo de esta actividad permitió mejorar nuestra utilización y percepción del tiempo para realizar las actividades y nuestra capacidad de manejo de excepciones. Además de la correcta implementación y análisis de los casos de prueba para las pruebas unitarias, mejorando la capacidad de implementar mejores casos. En resumen, la realización de esta actividad nos permitió mejorar en ámbito general, el pensamiento e implementación de código para la solución de problemas.