Caso Números Adyacentes

Integrantes: Sebastián Martínez, Joaquín Silva, Gerson Urrea.
Asignatura: Programación Orientada a Objetos.
Fecha: 22/09/23.

Introducción

El presente informe contendrá el desarrollo de la actividad “Caso números adyacentes”, este contiene la descripción del caso, la implementación del método principal, determinación de excepciones y diseño del control de estas, diseño e implementación de pruebas unitarias.

Descripción del caso

El caso planteado tiene como objetivo la implementación de un método capaz de determinar el producto de mayor valor entre los elementos contiguos de un arreglo de números enteros, finalmente este valor es mostrado en la terminal. Se adjunta una tabla con el tiempo estimado y el real para la realización de cada actividad.

Actividad	Tiempo estimado	Tiempo real
0	10 min	13 min
1	25 min	17 min
2	20 min	23 min
3	25 min	35 min
4	30 min	40 min

Análisis de Caso

1.- parámetros de entrada (y tipo de dato asociado)

Un arreglo de enteros de largo entre 2 y 20 de una dimensión, las entradas de este arreglo deben estar en un rango entre -1000 y 1000.

2.- valor de retorno

valor de retorno es un int

3. instrucciones (o pasos fundamentales, *no programe nada aún*)

Se le entregará al método productoAdyacente el arreglo de enteros para que sea analizado, se creará una variable llamada valor_maximo que tendrá como valor el menor posible de los productos (siendo este valor -1000000, suponiendo que -1000 y 1000 sean contiguos) , calculará el producto entre los elementos arreglo[i]*arreglo[i+1], comparando cada resultado con la variable valor_maximo, así actualizando su valor, para finalmente retornar el valor final de valor_maximo.

Hay que manejar una excepción para la última posición del arreglo “n”, ya que al hacer el producto con el “n+1” se saldrá de los bordes del arreglo

Implementación del Método

Verificación Funcionamiento del Método

Verifique que su método hace algo correcto, usando el caso de prueba *arreglo = {1, -4, 2, 2, 5, -1}* y verificando que la salida del método es efectivamente 10

Al momento de ingresar el arreglo especificado, el valor de retorno del método creado es efectivamente 10.

Determinación y diseño de pruebas unitarias

Casos de prueba:

1.- Arreglo con largo negativo

Probar que el método retorna un error `IndexOutOfBoundsException` al usar -1

2.- Arreglo con largo mayor a 20.

Probar que el método retorna un error `IndexOutOfBoundsException` índice fuera de los bordes.

3.- Arreglo con largo menor a 2.

Probar que el método retorna un error `IndexOutOfBoundsException` índice fuera de los bordes.

4.- Arreglo que funcione.

Probar que el método si sirve cuando el arreglo tiene largo entre 2 y 20

5.- Arreglo que de el valor correcto.

Probar que el código funciona

OBS: Al diseñar e implementar casos de pruebas para este su método productoAdyacentes(...) {...} considere que un input “ideal” tiene las siguientes propiedades:

- 2 ≤ arreglo.length ≤ 20
- -1000 ≤ arreglo[i] ≤ 1000

Determinación y control de excepciones

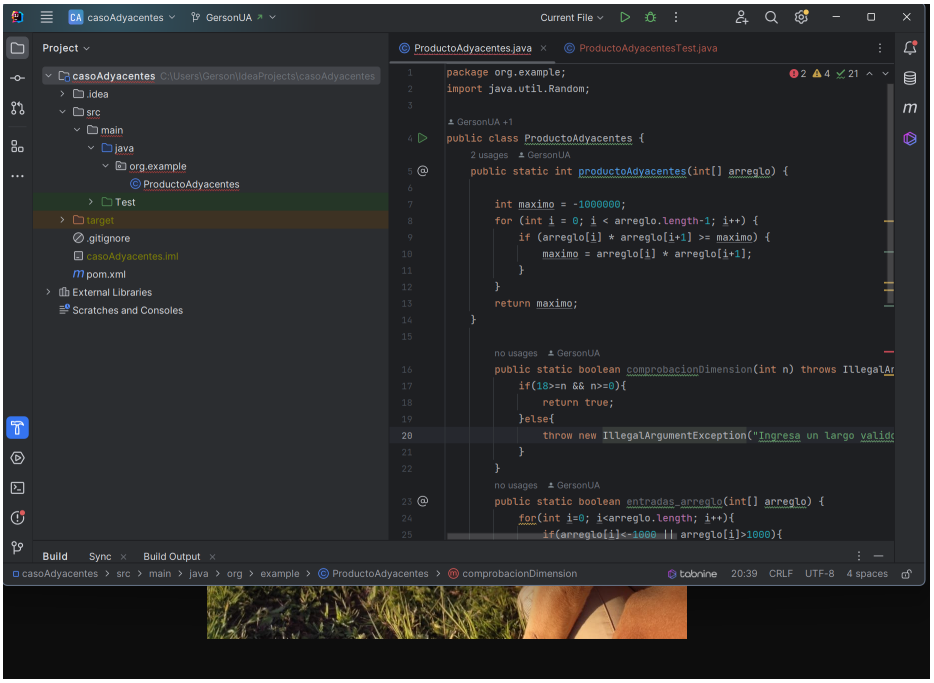
Errores Posibles y su control:

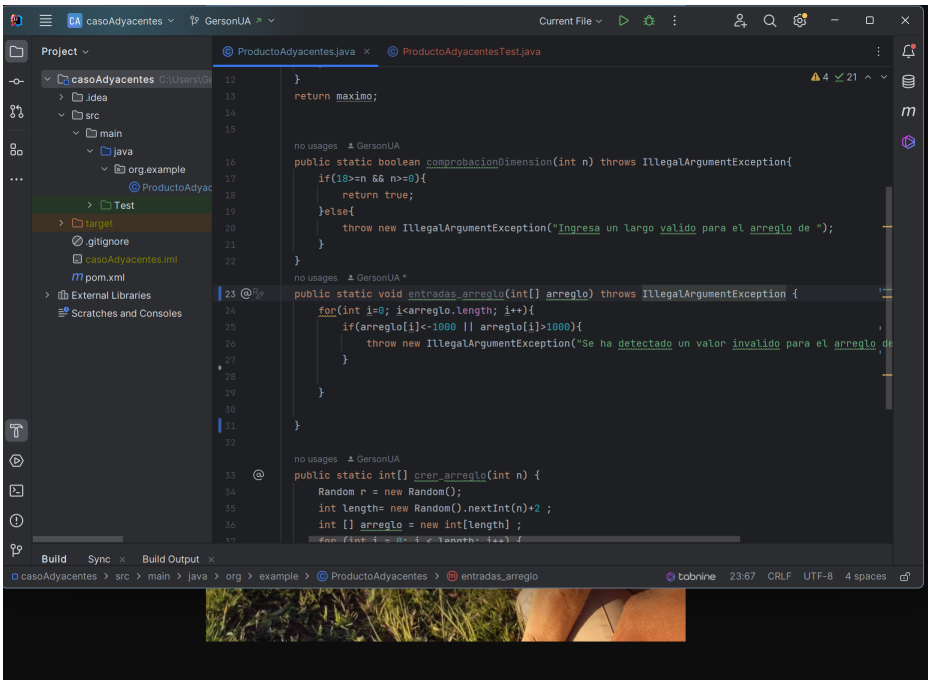
1. El Arreglo analizado de largo menor a 2 o mayor a 20.
   Implementar un método encargado de la validación del largo del arreglo a analizar, este método debe lanzar un error al detectar que el largo está fuera del rango establecido.
2. El arreglo analizado posee valores menores a -1000 o mayores a 1000.
   Implementar un método encargado de validar las entradas del arreglo ingresado, si detecta que existen entradas menores o mayores a estos valores se lanzará un error.
3.

Conclusión

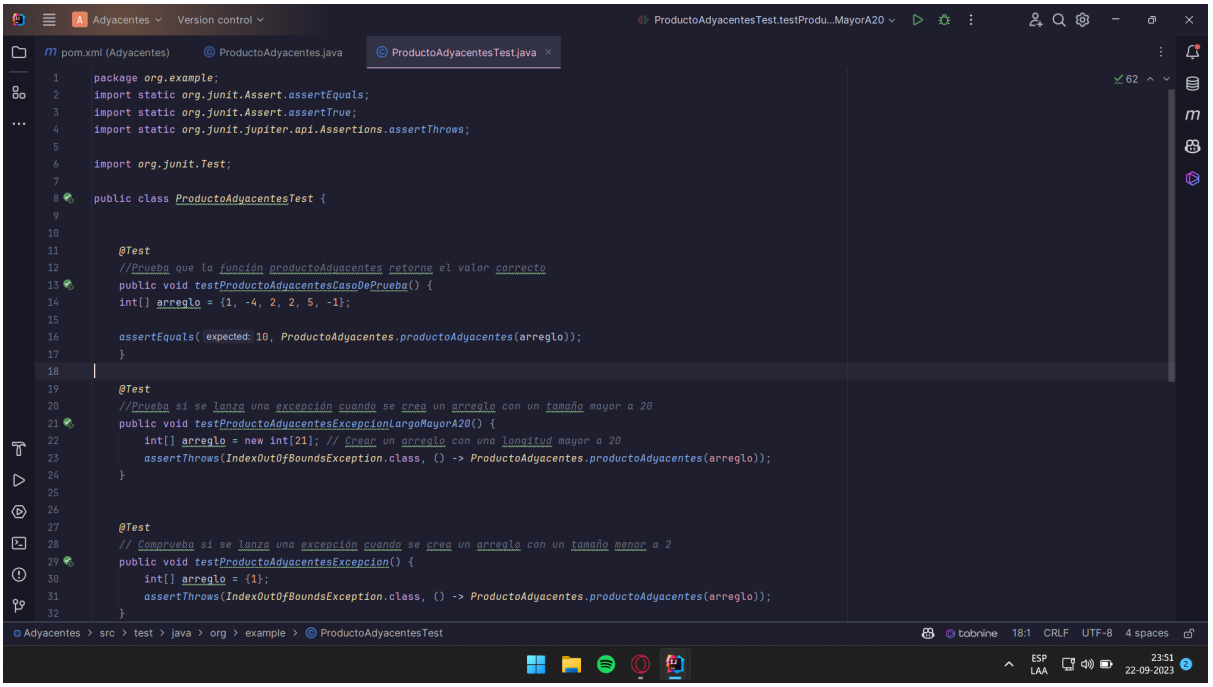
El análisis y desarrollo de este caso nos permitió practicar nuestras habilidades de análisis, interpretación y modelación de un problema. Este caso junto a los anteriores nos fueron de utilidad en el momento de la detección de excepciones y control del código creado, diseño e implementación de pruebas unitarias apropiadas para los métodos implementados y poner en práctica el diseño modular de soluciones a través de métodos conversado en clases. Para finalizar adjuntamos la evidencia de la contribución de cada uno de los integrantes.

Gerson Urrea

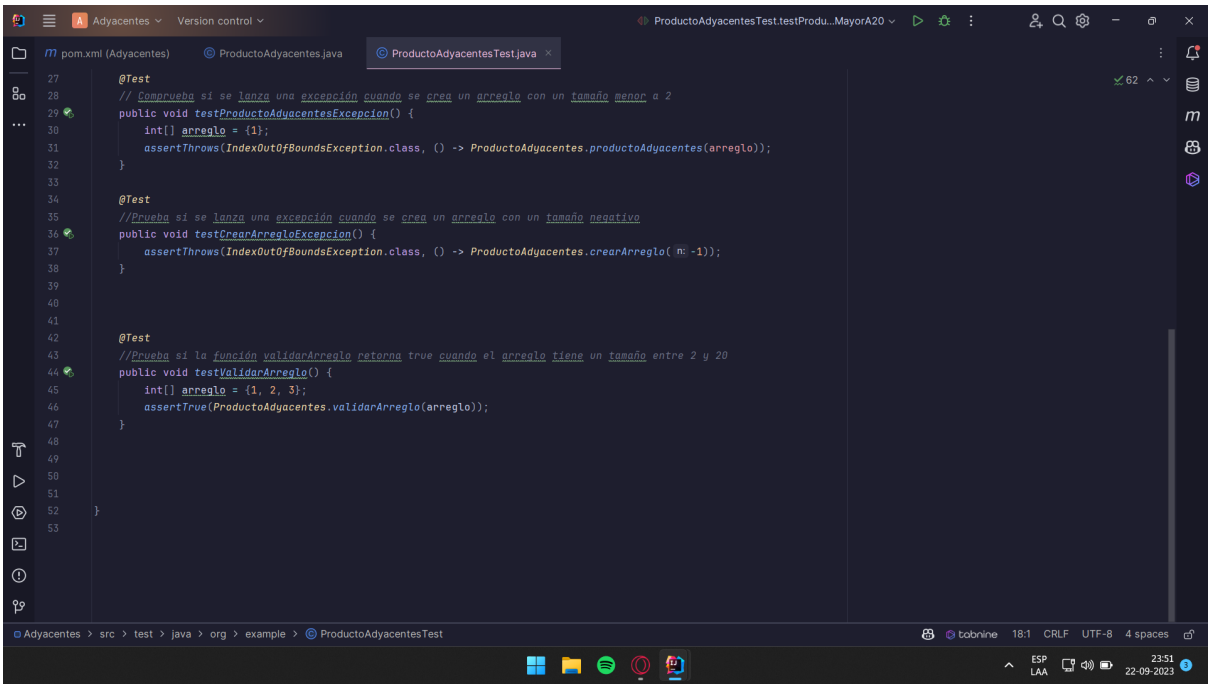




Sebastian Martinez



```java
package org.example;
import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertTrue;
import static org.junit.jupiter.api.Assertions.assertThrows;

import org.junit.Test;

public class ProductoAdyacentesTest {


    @Test
    //Prueba que la función productoAdyacentes retorne el valor correcto
    public void testProductoAdyacentesCasoDePrueba() {
    int[] arreglo = {1, -4, 2, 2, 5, -1};

    assertEquals( expected: 10, ProductoAdyacentes.productoAdyacentes(arreglo));
    }

    @Test
    //Prueba si se lanza una excepción cuando se crea un arreglo con un tamaño mayor a 20
    public void testProductoAdyacentesExcepcionLargoMayorA20() {
        int[] arreglo = new int[21]; // Crear un arreglo con una longitud mayor a 20
        assertThrows(IndexOutOfBoundsException.class, () -> ProductoAdyacentes.productoAdyacentes(arreglo));
    }


    @Test
    // Comprueba si se lanza una excepción cuando se crea un arreglo con un tamaño menor a 2
    public void testProductoAdyacentesExcepcion() {
        int[] arreglo = {1};
        assertThrows(IndexOutOfBoundsException.class, () -> ProductoAdyacentes.productoAdyacentes(arreglo));
    }
```



```java
    @Test
    // Comprueba si se lanza una excepción cuando se crea un arreglo con un tamaño menor a 2
    public void testProductoAdyacentesExcepcion() {
        int[] arreglo = {1};
        assertThrows(IndexOutOfBoundsException.class, () -> ProductoAdyacentes.productoAdyacentes(arreglo));
    }

    @Test
    //Prueba si se lanza una excepción cuando se crea un arreglo con un tamaño negativo
    public void testCrearArregloExcepcion() {
        assertThrows(IndexOutOfBoundsException.class, () -> ProductoAdyacentes.crearArreglo( n: -1));
    }



    @Test
    //Prueba si la función validarArreglo retorna true cuando el arreglo tiene un tamaño entre 2 y 20
    public void testValidarArreglo() {
        int[] arreglo = {1, 2, 3};
        assertTrue(ProductoAdyacentes.validarArreglo(arreglo));
    }



}
```

Joaquin Silva

```java
public class ProductoAdyacentes {

    public static void main(String[] args) {
        int[] arreglo = crearArreglo(2);
        int[] arreglo2 = {101,-1002};
        System.out.println(productoAdyacentes(arreglo));
        System.out.println(productoAdyacentes(arreglo2));
    }
    public static int[] crearArreglo(int n) {
        if (n >= 0){
            int[] arreglo = new int[n];
            for (int i = 0; i < arreglo.length; i++) {
                arreglo[i] = (int) (Math.random() * 2001 - 1000);
            }
            return arreglo;
        }else {
            throw new IndexOutOfBoundsException("El arreglo debe tener entre 2 y 20 elementos");
        }
    }
    public static boolean validarValores(int[] arreglo){
        for (int i = 0; i < arreglo.length; i++) {
            if (arreglo[i]<-1000 || arreglo[i]>1000){
                throw new IndexOutOfBoundsException("El arreglo debe tener valores que no sean menores a -1000 y mayores a 1000");
            }
        }return true;
    }
    public static boolean validarArreglo(int[] arreglo) {
        if (arreglo.length >=2 && arreglo.length<=20){
            return true;
        }else{
            return false;
```

```java
    public static boolean validarValores(int[] arreglo){
        for (int i = 0; i < arreglo.length; i++) {
            if (arreglo[i]<-1000 || arreglo[i]>1000){
                throw new IndexOutOfBoundsException("El arreglo debe tener valores que no sean menores a -1000 y mayores a 1000");
            }
        }return true;
    }
    public static boolean validarArreglo(int[] arreglo) {
        if (arreglo.length >=2 && arreglo.length<=20){
            return true;
        }else{
            return false;
        }
    }
    public static int productoAdyacentes(int[] arreglo) {
        int maximo = -1000000;
        if (validarArreglo(arreglo) && validarValores(arreglo)) {
            for (int i = 0; i < arreglo.length - 1; i++) {
                if (arreglo[i] * arreglo[i + 1] > maximo) {
                    maximo = arreglo[i] * arreglo[i + 1];
                }
            }
            return maximo;
        }else {
            throw new IndexOutOfBoundsException("El arreglo debe tener entre 2 y 20 elementos");
        }
    }
}
```