Universidad de la Frontera
Francisco Salazar 01145, Temuco, Araucanía
https://www.ufro.cl/

Caso de Números adyacentes Programación Orientada a Objetos

Integrantes:
Cristobal Rámos
Jonathan Chávez
Esteban Aguilera
Joaquín Arriagada

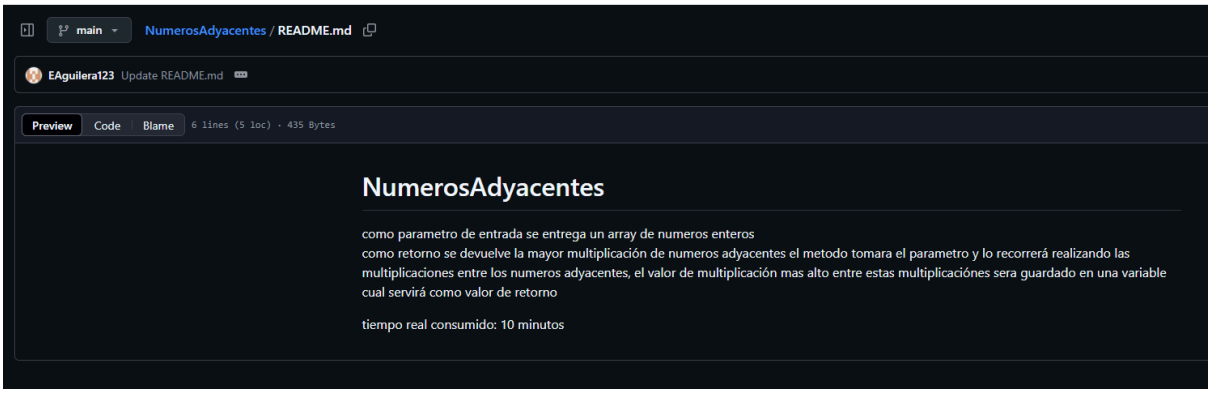
Introducción

Como equipo de trabajo se nos pide desarrollar una solución completa basada en métodos, que aplique las buenas prácticas al caso planteado. Además debemos considerar la implementación de pruebas unitarias usando Junit y la gestión de errores a través del uso de excepciones.

Descripción del caso

Se nos pide realizar la multiplicación a un arreglo de enteros, donde un método deberá retornar el mayor producto de números adyacentes que encuentre en ese arreglo.

0. Análisis de caso



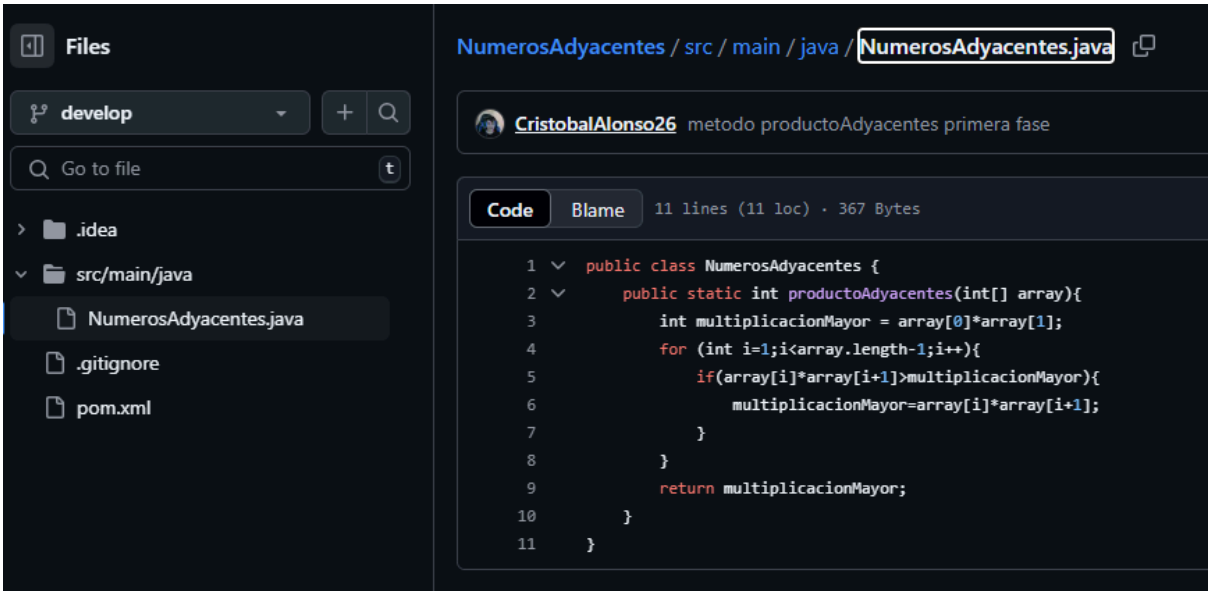
Se realizó un README en el repositorio de github con la descripción del caso junto al tiempo real utilizado para discutir el tema.

Números Adyacentes

Como parámetro de entrada se entrega un array de números enteros, como retorno se devuelve la mayor multiplicación de números adyacentes, el método tomará el parámetro y lo recorrerá realizando las multiplicaciones entre los números adyacentes, el valor de multiplicación más alto entre estas multiplicaciones será guardado en una variable cual servirá como valor de retorno
Tiempo real consumido: 10 minutos

Universidad de la Frontera
Francisco Salazar 01145, Temuco, Araucanía
https://www.ufro.cl/

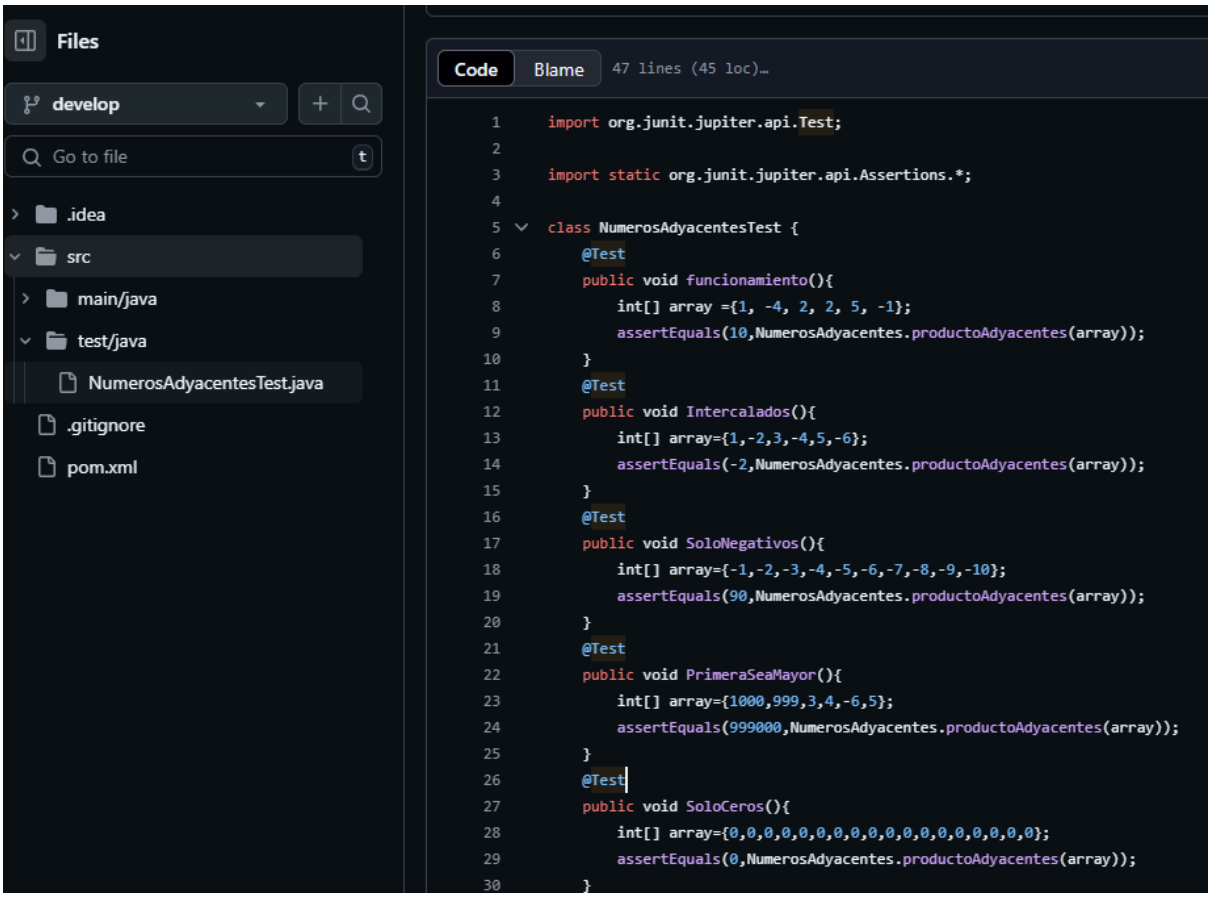
1. Implementación de la solución



Se realizó un push al repositorio con el método NumerosAdyacentes funcional
tiempo real consumido: 5 min

Universidad de la Frontera
Francisco Salazar 01145, Temuco, Araucanía
https://www.ufro.cl/

2. Diseño-implementación y resultados de las pruebas unitarias



Como casos iniciales pensamos en las siguientes situaciones de interés que se requieren verificar.

Caso 1: Arreglo con números intercalados entre positivos y negativos
Queremos probar que el método guarda el producto más alto dentro del arreglo (cercano a 0)

Caso 2: Arreglo con solo números negativos
Queremos probar que el método realmente hace la multiplicación de manera correcta

Caso 3: Arreglo con primera multiplicación alta
Queremos probar que el método guarda la primera multiplicación como la mayor si es que esta lo es

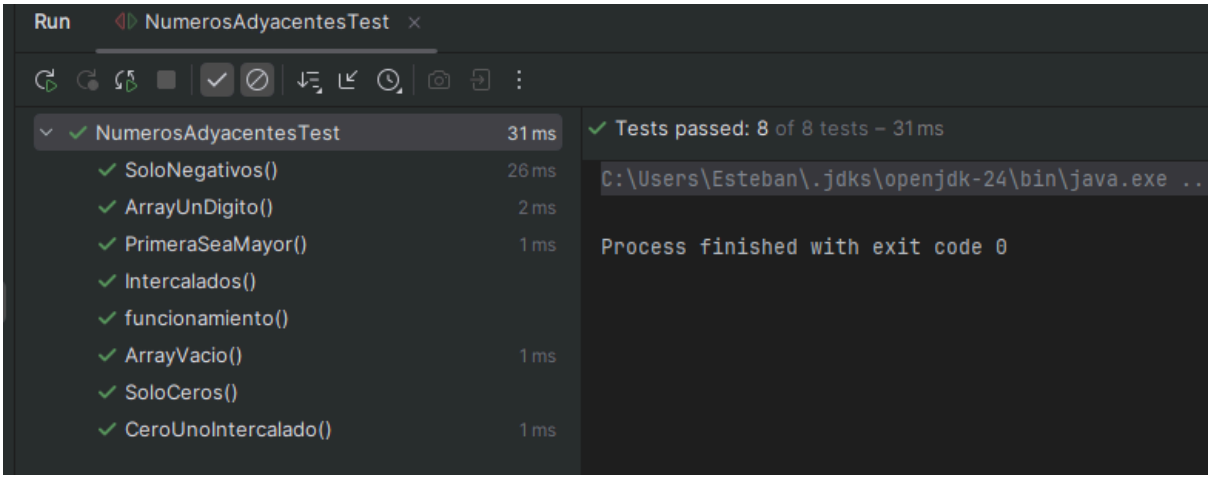
Caso 4: Arreglo con solo 0
Queremos probar que el método no falla al tener solo 0

Universidad de la Frontera
Francisco Salazar 01145, Temuco, Araucanía
https://www.ufro.cl/

Aquí se adjunta el código utilizado para alguno de los test unitarios.

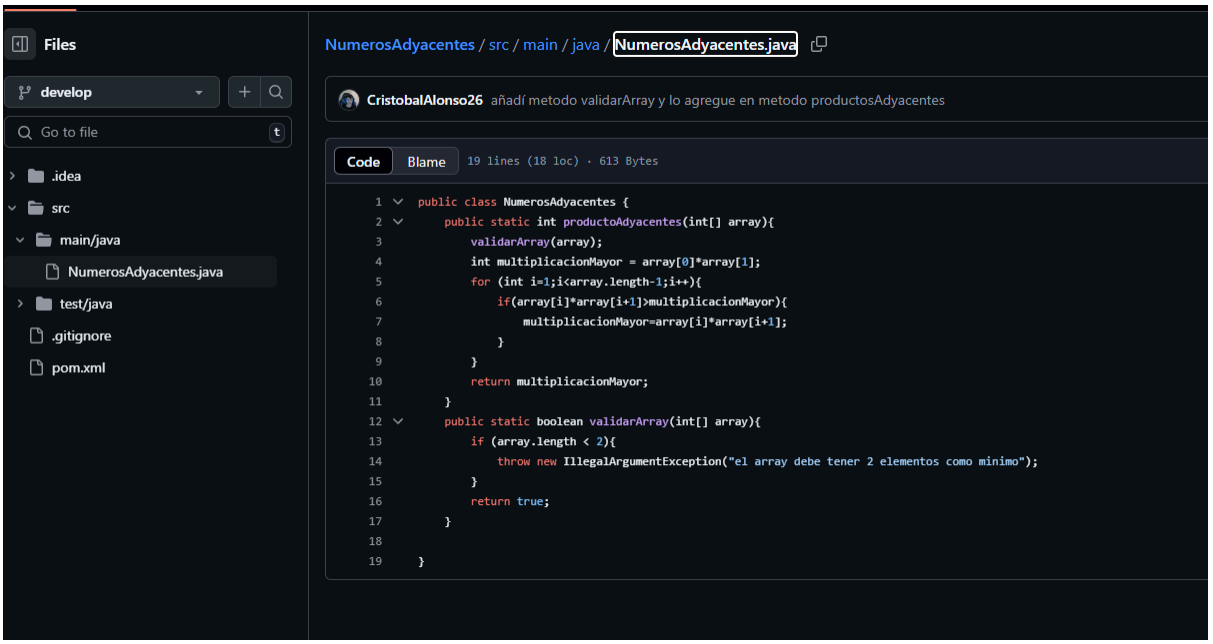
```
import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.*;

class NumerosAdyacentesTest {
    @Test
    public void funcionamiento(){
        int[] array ={1, -4, 2, 2, 5, -1};
        assertEquals(10,NumerosAdyacentes.productoAdyacentes(array));
    }
    @Test
    public void Intercalados(){
        int[] array={1,-2,3,-4,5,-6};
        assertEquals(-2,NumerosAdyacentes.productoAdyacentes(array));
    }
    @Test
    public void SoloNegativos(){
        int[] array={-1,-2,-3,-4,-5,-6,-7,-8,-9,-10};
        assertEquals(90,NumerosAdyacentes.productoAdyacentes(array));
    }
    @Test
    public void PrimeraSeaMayor(){
        int[] array={1000,999,3,4,-6,5};
        assertEquals(999000,NumerosAdyacentes.productoAdyacentes(array));
    }
}
```

Evidencia de pruebas superadas exitosamente



Universidad de la Frontera
Francisco Salazar 01145, Temuco, Araucanía
https://www.ufro.cl/

3. Diseño- implementación y resultados de las excepciones implementadas



Para garantizar el correcto funcionamiento de nuestro código, además de las pruebas unitarias debemos de crear un correcto manejo de excepciones en este, donde priorizamos los siguientes casos.

Para este paso definimos 3 tipos de errores:

Error 1: IndexOutOfBounds

Para evitar que el índice del arreglo saliera de límites nos aseguramos que mediante el bucle no se pueda acceder ni a elementos menores a 0 o mayores a la longitud del arreglo -1.

Error 2: IllegalArgument

Para capturar la excepción del caso que el arreglo tenga menos de 2 elementos agregamos el método de validarArray donde se capta la excepción.

Error 3: Asegurarse que la primera multiplicación se guardase

Para poder asegurarnos que la primera multiplicación entre el índice 0 y 1 se guardase de manera correcta, agregamos esa multiplicación como valor inicial de la variable multiplicacionMayor.

Con estos errores consideramos que el código es lo suficientemente robusto como para no fallar en su ejecución.

Universidad de la Frontera
Francisco Salazar 01145, Temuco, Araucanía
https://www.ufro.cl/

4. Discusión con los resultados y comentarios de la experiencia
Se nos presentó un problema acotado en el cual pudimos practicar el crear pruebas unitarias pertinentes y funcionales al problema, por esto mismo consideramos que fue una experiencia de aprendizaje apropiada.

5. Conclusiones.
Luego de hacer este taller pudimos concluir que la metodología donde se consideran las especificaciones técnicas del proyecto (hoja de requerimientos) para hacer tanto los métodos como tests expedita el debugging en caso de necesitarlo ya que cada problema y solución se atomiza.

Anexo link de GitHub utilizado.
https://github.com/CristobalAlonso26/NumerosAdyacentes