

Facultad de Ingeniería y Ciencias  
Departamento de Ciencias de la Computación e Informática



**Evidencia de la semana**  
Caso de los números adyacentes

**Integrantes**

Yasmín Hernández

Natalia Pérez

Catalina Quidel

Sophia Torres

**Fecha de entrega**

08/05/2025

## Introducción

El presente trabajo tiene como objetivo desarrollar una solución programática modular basada en métodos, que permita resolver de manera eficiente un problema computacional simple pero representativo: encontrar el mayor producto entre elementos adyacentes en un arreglo de enteros. A través de esta actividad, se busca aplicar buenas prácticas de desarrollo de software, como la separación de responsabilidades en métodos, la gestión adecuada de errores mediante excepciones, y la validación del funcionamiento del código mediante pruebas unitarias con JUnit.

El caso propuesto consiste en implementar un método llamado **productoAdyacentes**, el cual recibe como parámetro un arreglo de números enteros y retorna el mayor producto que se pueda obtener entre dos elementos consecutivos del mismo. A modo de ejemplo, si el arreglo es  $\{1, -4, 2, 2, 5, -1\}$ , el resultado esperado es 10, producto de multiplicar 2 y 5.

La actividad será desarrollada en un entorno de trabajo grupal guiado, con instancias de revisión por parte del docente o ayudante. Se utilizarán herramientas como IntelliJ IDEA para la implementación, GitHub para el control de versiones y JUnit para la creación de casos de prueba automatizados. Durante el desarrollo, se registrarán los tiempos reales de trabajo y se documentarán los avances con capturas de pantalla que evidencien el progreso.

Finalmente, se elaborará un reporte completo que documente el análisis del problema, las decisiones de diseño tomadas, la implementación del código, los resultados de las pruebas unitarias y la gestión de errores, así como una reflexión final sobre la experiencia del grupo en la resolución del caso.

## Objetivo

Diseñar e implementar un módulo funcional en lenguaje Java que resuelva el problema de detección del mayor producto entre elementos adyacentes en un arreglo unidimensional de enteros. La solución debe estar estructurada mediante una arquitectura modular que favorezca la mantenibilidad y la reutilización del código. Se requiere incorporar pruebas unitarias automatizadas utilizando el framework JUnit 5, con el fin de validar la corrección del algoritmo ante diferentes escenarios de entrada, incluyendo casos límite. Además, el método debe estar reforzado con mecanismos de control de errores mediante el uso explícito de excepciones, garantizando así un comportamiento predecible y robusto ante entradas inválidas. El proceso debe ser documentado y versionado utilizando herramientas estándar de la industria como IntelliJ IDEA y GitHub.

## Actividad 0. Análisis del caso

Durante esta etapa inicial, el grupo procederá a revisar y comprender detalladamente el enunciado del problema propuesto. Se realizará una estimación del tiempo requerido para su desarrollo y, a través de una discusión grupal, se definirá la estructura general del método `productoAdyacentes`. Se identificarán los parámetros de entrada, el tipo de valor de retorno y los pasos lógicos fundamentales del algoritmo. Este análisis permitirá orientar correctamente el diseño de la solución, asegurando una base conceptual sólida para las etapas siguientes.

- Tiempo estimado, para definir una estructura de método: 20 min.

Desarrollo:

**Parámetros del método `productoAdyacentes(...)`:**

- Parámetros de entrada:
  - `int[] arreglo` → un arreglo de números enteros.
- Valor de retorno:
  - `int` → el mayor producto encontrado entre dos números adyacentes.

**Pasos fundamentales (pseudocódigo):**

1. Verificar que el arreglo no sea nulo y tenga al menos dos elementos.
2. Inicializar una variable `maxProducto` con el primer producto adyacente.
3. Recorrer el arreglo desde el segundo elemento y calcular el producto del elemento actual con el anterior.
4. Comparar ese producto con `maxProducto` y actualizar si es mayor.
5. Al finalizar, retornar `maxProducto`.

## Actividad 1: implementar su método

### Desarrollo:

Tiempo estimado para la respuesta grupal: 45 minutos

Tiempo real consumido: 1 hora y 28 minutos

El código adjunto en captura , es la respuesta grupal después del consenso de los 4 códigos individuales.

El método estático `productoAdyacentes` toma como entrada un arreglo de enteros (`int[] arreglo`) y retorna el mayor producto encontrado entre todos los pares de números adyacentes dentro de dicho arreglo. El método incluye validaciones iniciales para asegurar que el arreglo no sea nulo y contenga al menos dos elementos. Inicializa el `maxProducto` con el producto del primer par de elementos y luego itera a través del resto del arreglo, comparando el producto de cada par adyacente con el `maxProducto`.

También ha sido actualizado el repositorio con este código en la rama “main”, para visualizarlo , es posible acceder por <https://github.com/cato-punk/NumerosAdyacentes.git>

```
ProductoAdyacentes.java x
1 public class ProductoAdyacentes { new *
2
3 // Método que calcula el mayor producto entre números adyacentes en el arreglo
4 public static int productoAdyacentes(int[] arreglo) { 1usage new *
5 // Validación básica
6 if (arreglo == null) {
7     throw new IllegalArgumentException("El arreglo no puede ser nulo.");
8 }
9 if (arreglo.length < 2) {
10     throw new IllegalArgumentException("Debe haber al menos dos elementos.");
11 }
12
13 // Inicializamos el máximo producto con el primer par de elementos
14 int maxProducto = arreglo[0] * arreglo[1];
15
16 // Recorremos desde el segundo elemento hasta el penúltimo
17 for (int i = 1; i < arreglo.length - 1; i++) {
18     int producto = arreglo[i] * arreglo[i + 1];
19     if (producto > maxProducto) {
20         maxProducto = producto; // actualizamos si encontramos un mayor producto
21     }
22 }
23
24 return maxProducto;
25 }
26
27 // Método main para probar el método
28 public static void main(String[] args) { new *
29     int[] arreglo = {1, -4, 2, 2, 5, -1};
30     System.out.println("El producto adyacente es: " + productoAdyacentes(arreglo));
```

## Actividad 2: Probando su código

En esta instancia, el equipo diseñó una serie de casos de prueba representativos para evaluar el correcto funcionamiento del método bajo distintas condiciones. Se incluyeron escenarios con números positivos, negativos, ceros, valores extremos y arreglos de tamaño mínimo. A partir de estos casos, se identificaron los valores esperados de retorno y se preparó el entorno de pruebas utilizando JUnit.

Caso 1 → Caso típico positivo

Arreglo: { 2, 5, -3, -2, 2, 7 }

Se debe verificar que el método está identificando de forma correcta el mayor producto entre adyacentes positivos.

Por lo tanto, el retorno es 14.

```
6      @Test
7      public void testCasoBase() {
8          int[] arreglo = {2, 5, -3, -2, 2, 7};
9          assertEquals( expected: 14, ProductoAdyacentes.productoAdyacentes(arreglo));
10     }
```

Caso 2 → Todos los elementos negativos.

Arreglo: {-2, -6, -10, -3}

Se verifica el funcionamiento cuando todos los elementos son negativos.

El retorno esperado es 60.

```
@Test
public void testNegativos() {
    int[] arreglo = {-2, -6, -10, -3};
    assertEquals( expected: 60, ProductoAdyacentes.productoAdyacentes(arreglo));
}
```

Caso 3 → Arreglo mínimo permitido.

Arreglo: {5}

Se corrobora el comportamiento en el caso límite inferior de longitud del arreglo.

```
@Test
public void testExcepcionPorTamaño() {
    int[] arreglo = {5};
    assertThrows(IllegalArgumentException.class, () -> {
        ProductoAdyacentes.productoAdyacentes(arreglo);
    });
}
```

Caso 4 → Todos los elementos son cero.

Arreglo: {0, 0, 0}

```
@Test
public void testTodosCeros() {
    int[] arreglo = {0, 0, 0};
    assertEquals(expected: 0, ProductoAdyacentes.productoAdyacentes(arreglo));
}
```

Caso 5 → Valores altos.

Se verifica si los valores altos son manejados correctamente.

```
@Test
public void testValoresAltos() {
    int[] arreglo = {1000, 1000};
    assertEquals(expected: 1_000_000, ProductoAdyacentes.productoAdyacentes(arreglo));
}
```

Caso 6 → Ceros en el arreglo que confundan la lógica

```
@Test  Catalina
public void testCerosConfundenLogica() {
    int[] arreglo = {0, 0, 0, -1000, -1000, 0, 0}; // -1000 * -1000 = 1_000_000
    String esperado = "El mayor producto adyacente es: -1000 * -1000 = 1000000";
    String obtenido = productosAdyacentes.productoAdyacentesSimple(arreglo);
    assertEquals(esperado, obtenido);
}
```

Caso 7 → Existencia de dos números altos , para corroborar que no solo trabaje con el primer número de alto valor.

```
@Test  Catalina
public void testProductoAdyacenteMayor() {
    int[] arreglo = {3, -2, 5, 3, -5, 4, 5, 2};
    String resultadoEsperado = "El mayor producto adyacente es: 4 * 5 = 20";
    String resultadoObtenido = productosAdyacentes.productoAdyacentesSimple(arreglo);
    assertEquals(resultadoEsperado, resultadoObtenido);
}
```

## Actividad 3: Garantizando el éxito

Durante esta fase, se tomaron en cuenta los potenciales errores que podrían ocurrir durante la ejecución del programa. Se analizaron escenarios como la recepción de un arreglo nulo, la presencia de menos de dos elementos, la existencia de valores que excedieran los límites definidos o el manejo de arreglos de gran tamaño. Para gestionar estas situaciones, se planificaron e implementaron mecanismos de control de errores a través del uso de excepciones (`IllegalArgumentException`)

Se definieron mensajes específicos para cada tipo de error, buscando que fueran claros para entender qué había pasado. Esta inclusión de validaciones se hizo para que el método fuera más resistente y no se rompiera con cualquier entrada rara.

### Desarrollo:

#### Posibles errores:

1. Arreglo nulo
  - Gestión: lanzar `IllegalArgumentException` con mensaje claro.
2. Arreglo con menos de 2 elementos
  - Gestión: lanzar `IllegalArgumentException`.
3. Valores extremos fuera de rango
  - Gestión: opcional validar que  $-1000 \leq \text{arreglo}[i] \leq 1000$  (aunque no es obligatorio si se confía en los datos).
4. Arreglo muy grande (performance)
  - Gestión: agregar validación de longitud máxima (por ejemplo 20).

```
ProductoAdyacentes.java x
1 public class ProductoAdyacentes {
2     public static int productoAdyacentes(int[] arreglo) { 1 usage
3         validarEntrada(arreglo);
4
5         int maxProducto = arreglo[0] * arreglo[1];
6         for (int i = 1; i < arreglo.length - 1; i++) {
7             int producto = arreglo[i] * arreglo[i + 1];
8             if (producto > maxProducto) {
9                 maxProducto = producto;
10            }
11        }
12        return maxProducto;
13    }
14    private static void validarEntrada(int[] arreglo) { 1 usage
15        if (arreglo == null) {
16            throw new IllegalArgumentException("El arreglo no puede ser null.");
17        }
18        if (arreglo.length < 2) {
19            throw new IllegalArgumentException("El arreglo debe tener al menos 2 elementos.");
20        }
21        if (arreglo.length > 20) {
22            throw new IllegalArgumentException("El arreglo no debe tener más de 20 elementos.");
23        }
24        for (int valor : arreglo) {
25            if (valor < -1000 || valor > 1000) {
26                throw new IllegalArgumentException("Los valores deben estar entre -1000 y 1000.");
27            }
28        }
29    }
30
31    public static void main(String[] args) {
32        int[] arreglo = {1, -4, 2, 2, 5, -1};
33        try {
34            int resultado = productoAdyacentes(arreglo);
35            System.out.println("El producto adyacente es: " + resultado);
36        } catch (IllegalArgumentException e) {
37            System.out.println("Error: " + e.getMessage());
38        }
39    }
}
```

# Test:

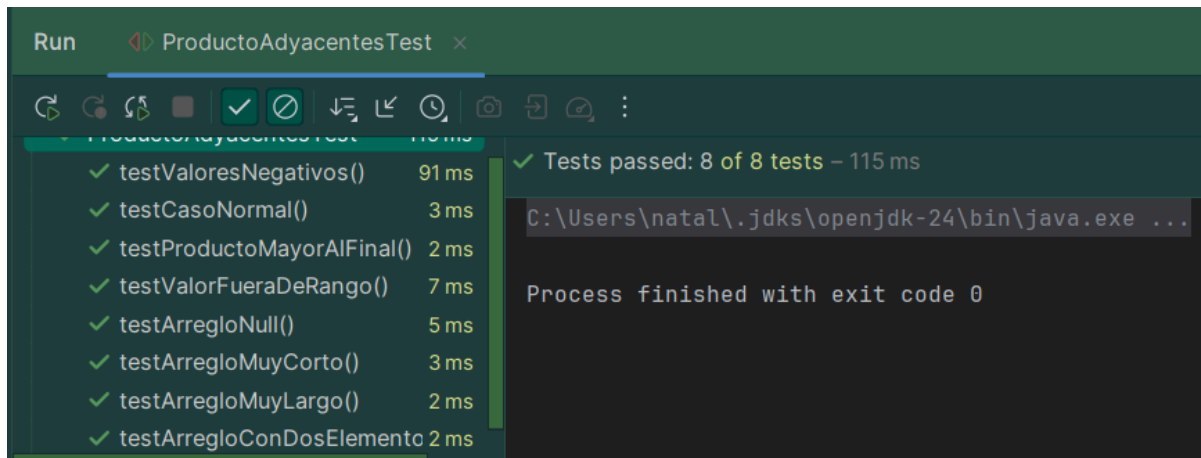
```
ProductoAdyacentes.java  ProductoAdyacentesTest.java x
1  import org.junit.jupiter.api.Test;
2  import static org.junit.jupiter.api.Assertions.*;
3
4  public class ProductoAdyacentesTest { new *
5
6      @Test new *
7      void testCasoNormal() {
8          int[] arreglo = {1, -4, 2, 2, 5, -1};
9          assertEquals( expected: 10, ProductoAdyacentes.productoAdyacentes(arreglo));
10     }
11
12     @Test new *
13     void testValoresNegativos() {
14         int[] arreglo = {-10, -20, -1, -2};
15         assertEquals( expected: 200, ProductoAdyacentes.productoAdyacentes(arreglo));
16     }
17
18     @Test new *
19     void testArregloConDosElementos() {
20         int[] arreglo = {3, 7};
21         assertEquals( expected: 21, ProductoAdyacentes.productoAdyacentes(arreglo));
22     }
23
24     @Test new *
25     void testProductoMayorAlFinal() {
26         int[] arreglo = {1, 2, 3, 4, 1000};
27         assertEquals( expected: 4000, ProductoAdyacentes.productoAdyacentes(arreglo));
28     }
29 }
```

```
ProductoAdyacentes.java  ProductoAdyacentesTest.java x
4  public class ProductoAdyacentesTest { new *
29
30      // Casos de error
31
32      @Test new *
33      void testArregloNull() {
34          IllegalArgumentException e = assertThrows(IllegalArgumentException.class, () ->
35              ProductoAdyacentes.productoAdyacentes( arreglo: null));
36          assertEquals( expected: "El arreglo no puede ser null.", e.getMessage());
37      }
38
39      @Test new *
40      void testArregloMuyCorto() {
41          int[] arreglo = {5};
42          IllegalArgumentException e = assertThrows(IllegalArgumentException.class, () ->
43              ProductoAdyacentes.productoAdyacentes(arreglo));
44          assertEquals( expected: "El arreglo debe tener al menos 2 elementos.", e.getMessage());
45      }
46
47      @Test new *
48      void testArregloMuyLargo() {
49          int[] arreglo = new int[21];
50          IllegalArgumentException e = assertThrows(IllegalArgumentException.class, () ->
51              ProductoAdyacentes.productoAdyacentes(arreglo));
52          assertEquals( expected: "El arreglo no debe tener más de 20 elementos.", e.getMessage());
53      }
54
55      @Test new *
56      void testValorFueraDeRango() {
57          int[] arreglo = {1001, 2};
```

```
54
55      @Test new *
56      void testValorFueraDeRango() {
57          int[] arreglo = {1001, 2};
58          IllegalArgumentException e = assertThrows(IllegalArgumentException.class, () ->
59              ProductoAdyacentes.productoAdyacentes(arreglo));
60          assertEquals( expected: "Los valores deben estar entre -1000 y 1000.", e.getMessage());
61      }
62  }
```



## Resultados:



```
Run  ProductoAdyacentesTest x
```

Test Name	Duration
✓ testValoresNegativos()	91 ms
✓ testCasoNormal()	3 ms
✓ testProductoMayorAlFinal()	2 ms
✓ testValorFueraDeRango()	7 ms
✓ testArregloNull()	5 ms
✓ testArregloMuyCorto()	3 ms
✓ testArregloMuyLargo()	2 ms
✓ testArregloConDosElemento	2 ms

✓ Tests passed: 8 of 8 tests – 115 ms

```
C:\Users\natal\.jdk\openjdk-24\bin\java.exe ...
```

Process finished with exit code 0

## Conclusión

A lo largo del desarrollo del caso "productoAdyacentes", el equipo logró implementar una solución robusta y modular que resuelve correctamente el problema propuesto. Se aplicaron buenas prácticas de programación al estructurar el código en métodos bien definidos, y se incluyó una adecuada gestión de errores mediante el uso de excepciones, lo que permitió cubrir distintos escenarios inesperados durante la ejecución.

Además, se diseñaron e implementaron pruebas unitarias con JUnit, las cuales permitieron validar el correcto funcionamiento del método bajo distintos casos de uso, tanto típicos como límite. Esto no solo aumentó la confiabilidad del programa, sino que también permitió identificar mejoras en la validación de entradas.

El uso de herramientas como IntelliJ IDEA, GitHub y JUnit facilitó un entorno de trabajo profesional, favoreciendo el versionamiento, la colaboración y la automatización de pruebas. Esta experiencia permitió al grupo afianzar conocimientos clave de desarrollo orientado a la calidad del software.

En resumen, el trabajo en equipo, la planificación por etapas y la revisión continua de resultados fueron factores clave para el éxito del proyecto, permitiendo cumplir con todos los objetivos propuestos y dejando una base sólida para afrontar desafíos similares en el futuro.