**ESCUELA COLOMBIANA DE INGENIERÍA**

**PROGRAMACIÓN ORIENTADA A OBJETOS**

**Diseño y Pruebas.**

**Interacción entre objetos.**

**2023-2**

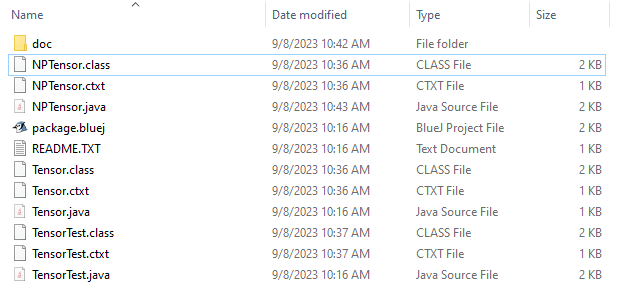
**Laboratorio 2/6**

1. **Conociendo el proyecto**

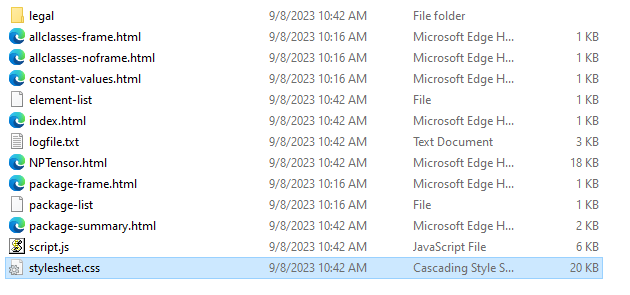
[En lab02.doc]

**1. El proyecto “TensorNP” contiene una construcción parcial del sistema. Revisen el directorio donde se encuentra el proyecto. Describan el contenido considerando los directorios y las extensiones de los archivos.**

2023-02-NPTensor\NPTensor



Nos encontramos con archivo de código compilado (NPTensor.class, Tensor.class, TensorTest.class), archivos de código fuente(NPTensor.java, Tensor.java, TensorTest.java), archivos de texto que contienen información adicional sobre los archivos Java (NPTensor.ctxt, Tensor.ctxt, TensorTest.ctxt), archivo de proyecto (package.bluej), y la carpeta doc *2023-02-NPTensor\NPTensor\do*c que almacena la documentación (.html).



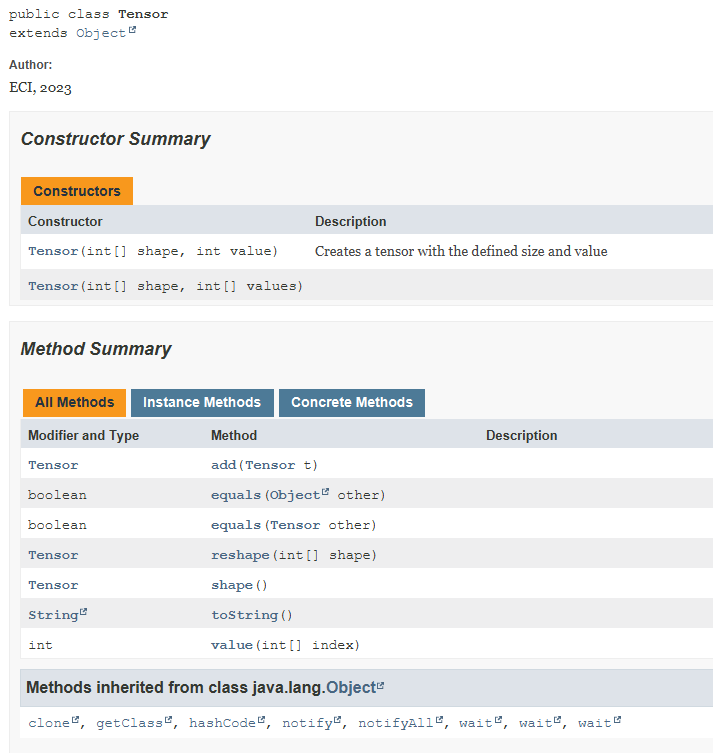
**2. Exploren el proyecto en BlueJ ¿Cuántas clases tiene? ¿Cuál es la relación entre ellas? ¿Cuál es la clase principal de la aplicación? ¿Cómo la reconocen? ¿Cuáles son las clases “diferentes”? ¿Cuál es su propósito?**

**Para las siguientes dos preguntas sólo consideren las clases “normales”**:

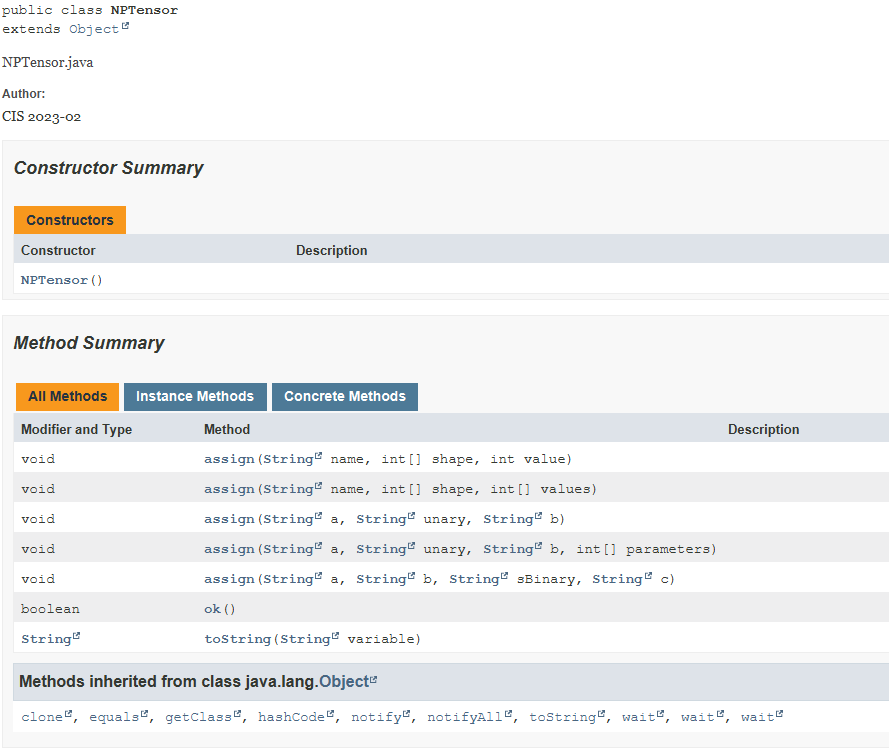
Tiene 3 clases, la relación consiste en que NPTensor y TensorTest utilizan metodos de Tensor. La clase principal de la aplicación es NPTensor, se reconoce porque es con la que se interactua para usar la aplicación. Las clase diferente es TensorTest, su proposito es realizar las pruebas unitarias de Tensor.

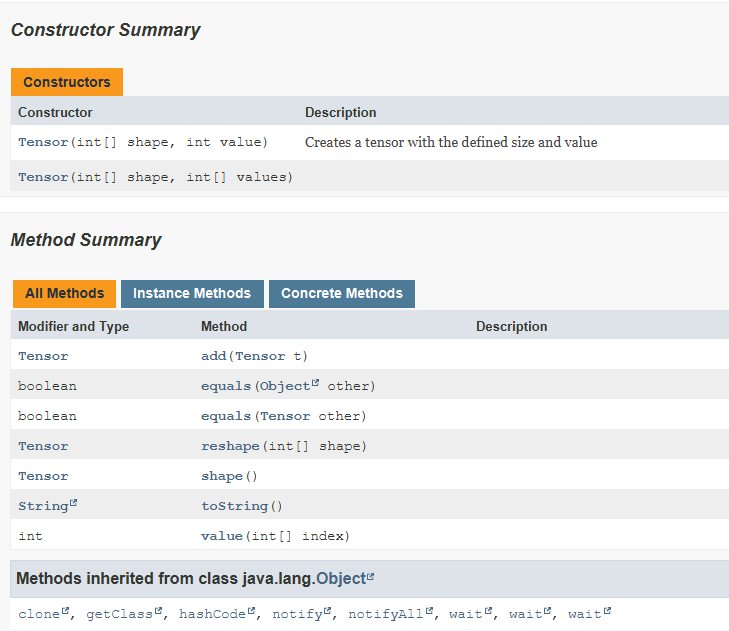
**3. Generen y revisen la documentación del proyecto: ¿está completa la documentación de cada clase? (Detallen el estado de documentación de cada clase: encabezado y métodos)**

La clase tensor tiene la documentación incompleta, pues solo cuenta con la descripción de lo que hace uno de sus constructores. Hace falta la documentación respectiva de cada método.



La clase NPTensor tiene la documentación incompleta, pues no tiene la documentación de los constructores y un comentario general de lo que hace la clase. Hace falta la documentación respectiva de cada método. Sin embargo, sus métodos tienen comentarios que aportan.





**4. Revisen las fuentes del proyecto, ¿en qué estado está cada clase? (Detallen el estado de las fuentes considerando dos dimensiones: la primera, atributos y métodos, y la segunda, código, documentación y comentarios) ¿Qué son el código, la documentación y los comentarios?**

* ***Clase NPTENSOR:***

Atributos: Tiene un atributo privado llamado variables, que es un HashMap destinado a almacenar tensores asociados a nombres de variables.

Métodos: Define varios métodos públicos para asignar tensores a variables y realizar operaciones en ellas. assign para asignar tensores, toString para obtener una representación en cadena de un tensor, y ok para verificar si la última operación fue exitosa.

Código: El código de la clase está incompleto. Los métodos assign tienen nombres descriptivos, pero su implementación está vacía. No se proporciona una implementación real de las operaciones que se describen en los comentarios. Además, se utiliza una referencia a la clase Tensor sin su definición, lo que hace que el código no sea funcional en su forma actual.

Documentación y Comentarios: La clase tiene algunos comentarios que describen brevemente los métodos y su propósito, pero no proporcionan detalles sobre cómo deben implementarse. La documentación es escasa y no se incluye información sobre los parámetros o los valores de retorno de los métodos. Se necesita una documentación más completa para entender el propósito y el funcionamiento de la clase.

* ***CLASE TENSOR****:*

Atributos: No se ven atributos definidos en esta clase.

Métodos: La clase Tensor define varios métodos públicos. La implementación real de estos métodos no está proporcionada en el código.

* Tensor(int[] shape, int value): Constructor que crea un tensor con un tamaño definido y un valor inicial.
* Tensor shape(): Método que devuelve un nuevo tensor con la forma (shape) del tensor actual.
* Tensor(int[] shape, int[] values): Constructor que crea un tensor con un tamaño definido y valores iniciales.
* int value(int[] index): Método que devuelve el valor en una ubicación específica del tensor según el índice proporcionado.
* Tensor reshape(int[] shape): Método que devuelve un nuevo tensor con una forma (shape) diferente.
* Tensor add(Tensor t): Método que realiza una operación de suma con otro tensor t y devolver el resultado como un nuevo tensor.
* boolean equals(Tensor other): Método que compara dos tensores y devuelve true si son iguales y false en caso contrario. La implementación real de este método no está proporcionada en el código.
* boolean equals(Object other): Sobrecarga del método equals para aceptar un objeto genérico y luego llamar al método equals(Tensor other) para realizar la comparación.
* String toString(): Método que devuelve una representación en cadena del tensor.

Código: La implementación de los métodos está incompleta en su forma actual. Todos los métodos devuelven valores nulos o valores por defecto, dándonos a entender que la clase Tensor no realiza ninguna funcionalidad real en su estado actual.

Documentación y Comentarios: La clase Tensor tiene una documentación básica (solo el constructor está), no hay comentarios en los métodos y no proporcionan detalles de implementación. La documentación proporciona una breve descripción de la clase, pero no incluye detalles sobre cómo se deben usar los métodos ni qué hacen exactamente. Los comentarios en los métodos son escasos.

**Código**: El código es el conjunto de instrucciones que se utilizan para crear un programa informático.

**Documentación**: La documentación es un conjunto de textos que explican el funcionamiento de un programa informático.

**Comentarios**: Los comentarios son textos que se añaden al código para explicar su funcionamiento.

1. **Ingeniería reversa**

[En lab02.doc TensorNP.asta]

**MDD MODEL DRIVEN DEVELOPMENT**

**1. Complete el diagrama de clases correspondiente al proyecto (No incluya la clase de pruebas)**

En Astah.

**2. ¿Cuál nuevo contenedor está definido? Consulte la especificación y el API Java ¿Qué diferencias hay entre el nuevo contenedor, el ArrayList y el vector [] que conocemos?**

El nuevo contenedor que está definido es HashMap.

La diferencia entre HashMap, ArrayLisy y vector[] es que en cuanto a tipo de estructura el HashMap es una colección de pares clave-valor, mientras que el ArrayList es una lista con un tamaño dinámico y el vector [] es una lista con un tamaño fijo. Por otro lado la indexación del HashMap es por clave, mientras que el ArrayList y el vector [] es por índice.

1. **Conociendo Pruebas en BlueJ**

[En lab02.doc \*.java]

***De TDD → BDD (TEST → BEHAVIOUR DRIVEN DEVELOPMENT)***

Para poder cumplir con las prácticas XP vamos a aprender a realizar las pruebas de unidad usando las herramientas apropiadas. Para eso consideraremos implementaremos algunos métodos en la clase **TensorTest.**

**1. Revisen el código de la clase TensorTest. ¿cuáles etiquetas tiene (componentes con símbolo @)? ¿cuántos métodos tiene? ¿cuantos métodos son de prueba? ¿cómo los reconocen?**

**Etiquetas**

@BeforeClass

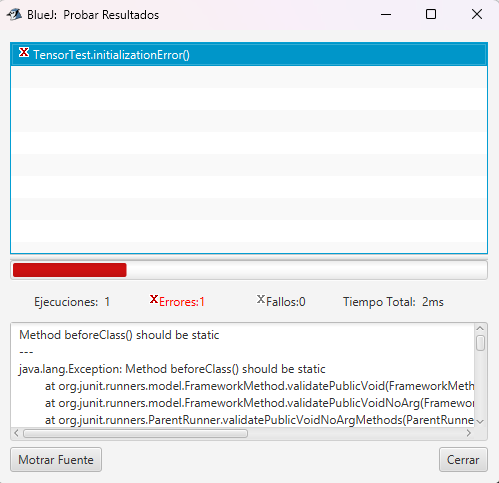
@Before

@Test

**Métodos**

Tiene 4, todos marcados como métodos de prueba y se reconocen con @Test.

**2. Ejecuten los tests de la clase TensorTest. (click derecho sobre la clase, Test All) ¿cuántas pruebas se ejecutan? ¿Cuántas pasan? ¿por qué?**



Se ejecutó una, pero no pasa porque esta presenta un error.

**3. Estudie las etiquetas encontradas en 1. Expliquen en sus palabras su significado.**

para controlar la ejecución de pruebas.

* @BeforeClass se ejecuta una vez antes de que se ejecute cualquier prueba en la clase.
* @Before se ejecuta antes de que se ejecute cada prueba (método) en la clase.
* @Test se utiliza para anotar un método que es una prueba unitaria.

**4. Estudie los métodos assertTrue, assertFalse, assertEquals, assertArrayEquals,assertNull y fail de la clase assert del API JUnit 2. Explique en sus palabras que hace cada uno de ellos.**

* **assertTrue:** Comprueba que una expresión booleana es verdadera. Si la expresión es falsa, la prueba fallará y se informará de un fallo.
* **assertFalse:** Comprueba que una expresión booleana es falsa. Si la expresión es verdadera, la prueba fallará y se informará de un fallo.
* **assertEquals:** Afirma que los dos valores especificados son iguales. Si los valores no son iguales, se lanza una excepción AssertionError.
* **assertArrayEqual:** afirma que los dos arreglos especificados son iguales. Si los arreglos no son iguales, se lanza una excepción AssertionError
* **assertNull :** Comprueba que una referencia a un objeto es nula. Si la referencia no es nula, la prueba fallará y se informará de un fallo.
* **fail:** hace que la prueba falle sin proporcionar ninguna información adicional.

**5. Investiguen la diferencia que entre un fallo y un error en Junit. Escriba código, usando los métodos del punto 4., para lograr que los siguientes tres casos de prueba se comporten como lo prometen shouldPass, shouldFail, shouldErr.**

Un fallo es una condición que se esperaba que sea verdadera pero que resultó ser falsa. Un error es una condición inesperada que ocurre durante la ejecución de una prueba.

public class Pruebas {

@Test

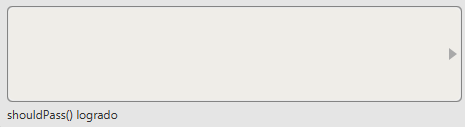
public void shouldPass() {

int a = 10;

int b = 10;

assertEquals(a, b);

}



@Test

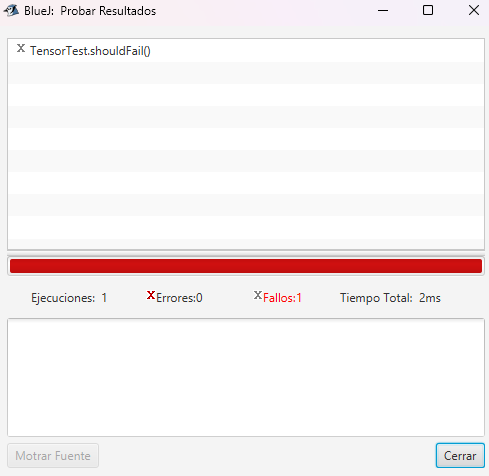
public void shouldFail() {

int a = 10;

int b = 20;

assertEquals(a, b);

}

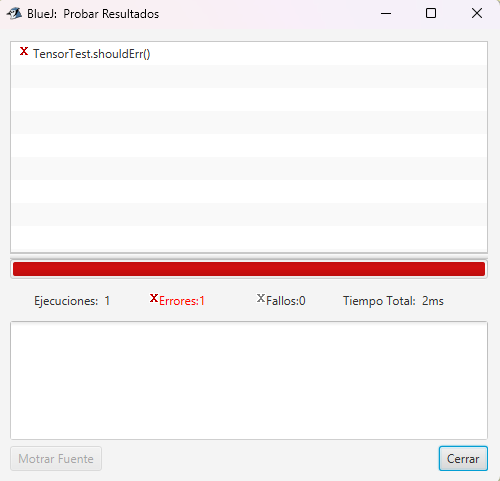


@Test

public void shouldErr() {

int result = 10 / 0; // Esto causará una excepción de división por cero

}



}

1. **Prácticando Pruebas en BlueJ**

[En lab02.doc \*.java]

**De TDD → BDD (TEST → BEHAVIOUR DRIVEN DEVELOPMENT)**

Ahora vamos escribir el código necesario para que las pruebas de TensorTest pasen.

**1. Determinen los atributos de la clase Tensor. Justifique la selección.**

* shape(int[])
* values(int[])  
    
  shape: Un arreglo de enteros que almacena la forma del tensor.

values: Un arreglo de enteros que almacena los valores del tensor.

Estos atributos se inicializan en los constructores y se utilizan en los métodos de la clase para realizar operaciones y acceder a los datos del tensor.

**2. Determinen el invariante de la clase Tensor. Justifique la decisión.**

La longitud del arreglo shape debe ser coherente con la cantidad de dimensiones de los datos en el arreglo values. Esto asegura que la forma del tensor corresponde con los datos que contiene. Si los tamaños de la forma y los valores no son coherentes, las operaciones en el tensor pueden resultar en errores o resultados incorrectos.

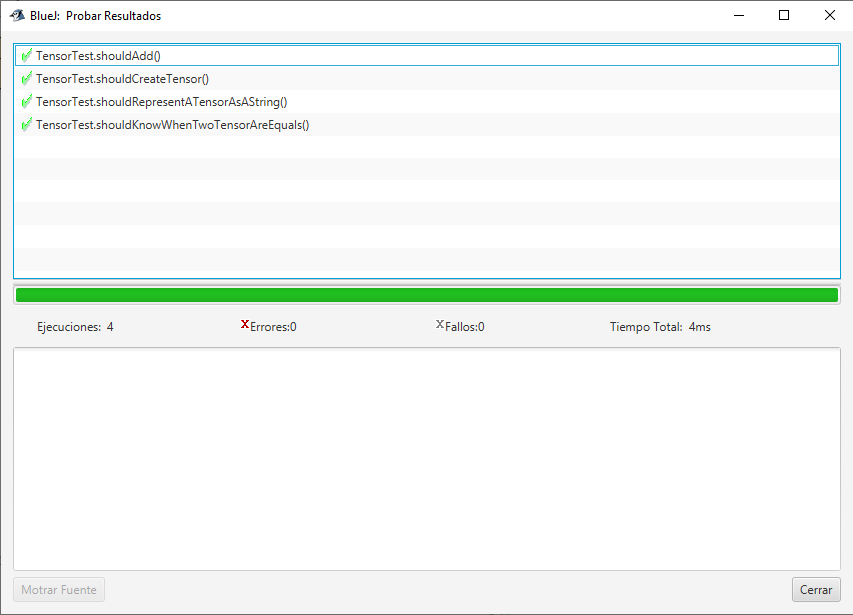
También el atributo shape siempre debe ser un arreglo de enteros no negativos, ya que representa las dimensiones del tensor.

**3. Implementen únicamente los métodos de Tensor necesarios para pasar todas las pruebas definidas. ¿Cuáles métodos implementaron?**

Los métodos utilizados para pasar todas la pruebas fueron:

* Tensor(int[] shape, int value)
* Tensor(int[] shape, int[] values)
* value(int[] index)
* Tensor reshape(int[] newShape)
* Tensor add(Tensor t)
* equals(Tensor other)
* equals(Object other)
* String toString()
* String buildString(Tensor tensor, int level)
* calculateSize(int[] shape)
* calculateFlatIndex(int[] index)

**4. Capturen los resultados de las pruebas de unidad.**



1. **Desarrollando TensorNP**

[En lab02.doc, TensorNP.asta, \*.java]

**BDD - MDD**

Para desarrollar esta aplicación vamos a considerar algunos mini-ciclos. En cada mini-ciclo deben realizar los pasos definidos a continuación.

**1.** Definir los métodos base correspondientes al mini-ciclo actual.

**2.** Definir y programar los casos de prueba de esos métodos

(piense en los deberia y los noDeberia (should and shouldNot)

**3.** Diseñar los métodos

(Use diagramas de secuencia. En astah, adicione el diagrama al método)

**4.** Escribir el código correspondiente (no olvide la documentación)

**5.** Ejecutar las pruebas de unidad (vuelva a 3 (a veces a 2), si no están en verde)

**6.** Completar la tabla de clases y métodos. (Al final del documento)

**Ciclo 1 :** Operaciones básicas de tensores: declarar, asignar un valor y consultar

**Ciclo 2 :** Operaciones unarias sin parámetros: dimensiones, redimensionar y barajar

**Ciclo 3 :** Operaciones unarias con parámetros: rebanada, media por ejes, find

**Ciclo 4 :** Operaciones binarias uno a uno: suma, resta, multiplicación

**BONO Ciclo 5 :** Defina tres nueva funcionalidades.

**Mini-ciclo1:**

1. - assign(String name, int[] shape, int value)

- assign(String name, int[] shape, int[] values)

- toString(String variable)

- ok()

- calculateSize(int[] shape)

- getValue(String name)

En Astah

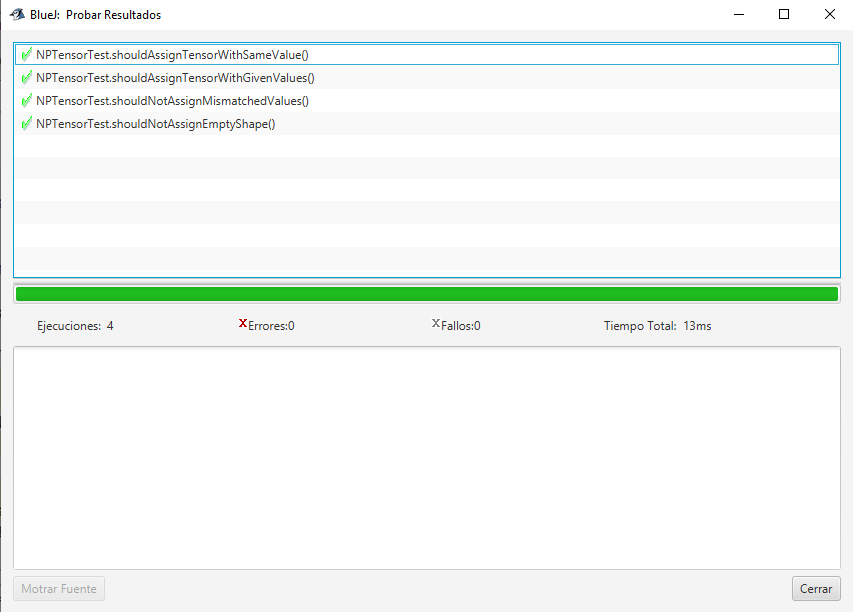
1. - shouldAssignTensorWithSameValue()

-shouldAssignTensorWithGivenValues()

-shouldNotAssignMismatchedValues()

-shouldNotAssignEmptyShape()

1. En Astah
2. En BlueJ



**Miniciclo 2:**

1. - assign(String a, String unary, String b)

En Astah

1. - testAssignShapeOperation()

- shouldReshapeTensor()

- shouldAssignShuffledTensor()

- shouldNotAssignInvalidUnaryOperation()

1. En Astah
2. En BlueJ

**Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente**

**Miniciclo 3:**

1. assign(String a, String unary, String b, int[] parameters)

En Astah

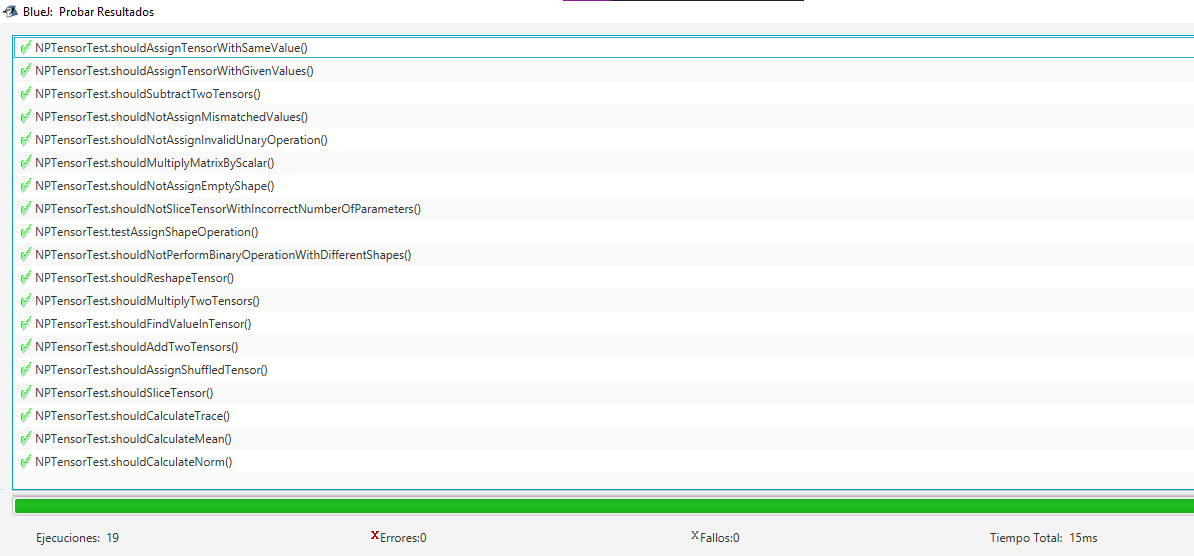
1. **-**shouldSliceTensor()

**-** shouldCalculateMean()

**-** shouldFindValueInTensor()

-shouldNotSliceTensorWithIncorrectNumberOfParameters()

1. En Astah
2. En BlueJ

****

**Miniciclo 4:**

1. **-** assign(String a, String b, String sBinary, String c)

En Astah

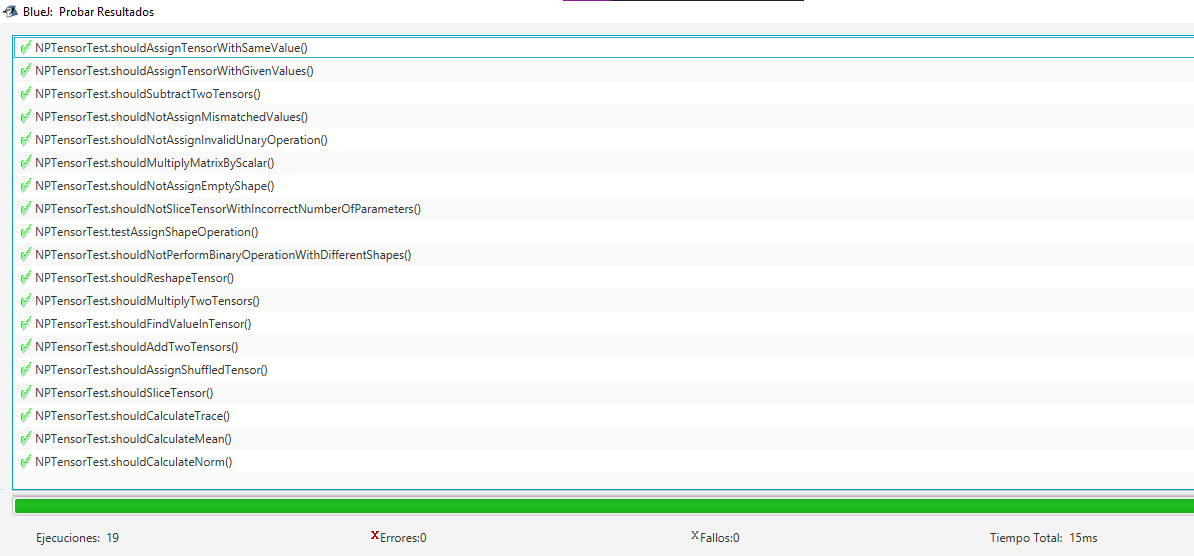
1. -shouldAddTwoTensors()

- shouldSubtractTwoTensors()

- shouldMultiplyTwoTensors()

-shouldNotPerformBinaryOperationWithDifferentShapes()

1. En Astah
2. En BlueJ



**Miniciclo 5:**

1. **-** calculate(String a, String b, String function)

- multiplyScalar(String a, String b, int scalar)

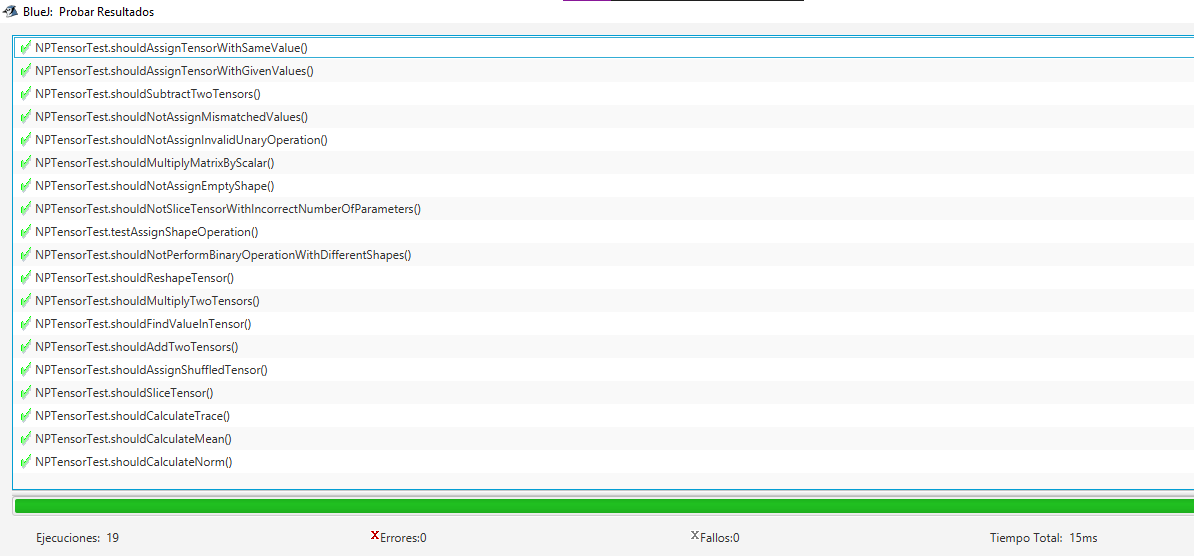
En Astah

1. -shouldCalculateNorm()

shouldCalculateTrace()

shouldMultiplyMatrixByScalar()

1. En Astah
2. En BlueJ



**6.**

|  |  |  |
| --- | --- | --- |
| **Mini-Ciclo** | **NPTensor** | **TensorNPTest** |
| **Ciclo 1** | - assign(String name, int[] shape, int value)  - assign(String name, int[] shape, int[] values)  - toString(String variable)  - ok()  - calculateSize(int[] shape)  - getValue(String name) | -shouldAssignTensorWithSameValue()  -shouldAssignTensorWithGivenValues()  -shouldNotAssignMismatchedValues()  -shouldNotAssignEmptyShape() |
| **Ciclo 2** | - assign(String a, String unary, String b) | - testAssignShapeOperation()  - shouldReshapeTensor()  -shouldAssignShuffledTensor()  -shouldNotAssignInvalidUnaryOperation() |
| **Ciclo 3** | -assign(String a, String unary, String b, int[] parameters) | -shouldSliceTensor()  - shouldCalculateMean()  - shouldFindValueInTensor() -shouldNotSliceTensorWithIncorrectNumberOfParameters() |
| **Ciclo 4** | - assign(String a, String b, String sBinary, String c) | -shouldAddTwoTensors()  - shouldSubtractTwoTensors()  - shouldMultiplyTwoTensors()  -shouldNotPerformBinaryOperationWithDifferentShapes() |
| **Ciclo 5** | calculate(String a, String b, String function)  multiplyScalar(String a, String b, int scalar) | shouldCalculateNorm()  shouldCalculateTrace()  shouldMultiplyMatrixByScalar() |

**RETROSPECTIVA**

1. **¿Cuál fue el tiempo total invertido en el laboratorio por cada uno de ustedes? (Horas/Hombre)**

(Ana María Durán/25), (Laura Natalia Rojas/ 25)

1. **¿Cuál es el estado actual del laboratorio? ¿Por qué?**

Abarcamos todos los puntos y los desarrollamos en su totalidad.

1. **Considerando las prácticas XP del laboratorio. ¿cuál fue la más útil? ¿por qué?**

All code must have unit tests, ya que aseguran la calidad del codigo al detectar errores a medida que se codifica.

1. **¿Cuál consideran fue el mayor logro? ¿Por qué?**

Haber logrado que los métodos funcionaran con sus respectivas pruebas unitarias.

1. **¿Cuál consideran que fue el mayor problema técnico? ¿Qué hicieron para resolverlo?**

Entender la funcionalidad de slice, y también el cómo debemos interpretar que funciona reshape en la prueba unitaria.

1. **¿Qué hicieron bien como equipo? ¿Qué se comprometen a hacer para mejorar los resultados?**

Trabajamos a la par, resolvimos dudas entre nosotras y buscamos ayuda cuando lo llegamos a necesitar. Nos comprometemos a seguir trabajando en equipo.