**Escuela Colombiana de Ingeniería Julio Garavito**

**LABORATORIO NUMERO. 02**

**Nombres:** Camilo Murcia Espinosa, Jeisson Casallas

9/8/2023

**Conociendo el proyecto**

**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.**

* Un archivo README, el cual deberia contener toda la información del proyecto en Bluej, sin embargo se encuentra vacio.
* Package.bluej, que es el ejecutable en donde podemos ver nuestro proyecto en Bluej

**Por cada una de las clases definidas en el Proyecto contiene lo siguiente:**

* Un archivo con extensión .class, que es el archive compilado de la clase el cual es el que podemos evidenciar al ejecutar el BlueJ.
* Un archive con extensión .ctxt, que contiene información adicional, como la documentación y los comentarios que se hacen acerca de cada clase y sus metodos.

* Un archive con extensión .java, que es el que contiene el codigo fuente de la clase.

**2. Exploren el proyecto en BlueJ**

**¿Cuántas clases tiene?**

Tiene 2 clases, que son NPTensor, Tensor.

**¿Cuál es la relación entre ellas?**

Las clase NPTensor necesita de la clase Tensor para poder usarse, por lo que una depende de la otra.

**¿Cuál es la clase principal de la aplicación? ¿Cómo la reconocen?**

La clase principal es NPTensor, ya que es la clase que ofrece los servicios de la aplicación. En este caso la clase que permite almacenar y operar tensores.

**¿Cuáles son las clases “diferentes”? ¿Cuál es su propósito?**

La clase diferente tiene el nombre TensorTest, su propósito es realizar pruebas sobre la clase Tensor, para verificar que esta cumple con todos sus requisitos, y realiza su trabajo de manera correcta.

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

**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)**

No, la documentacion esta incompleta, ya que solo se tiene un pequeño comentario sobre el constructor, pero no se especifican los parametros que entran en este, por otra parte, los demas metodos no tienen ninguna documentacion.

La clase de NPTensor esta igualmente incompleta y no cuenta con toda la documentacion correspondiente, ni en su encabezado ni en los metodos.

La clase de TensorTest tampoco esta documentada de forma correcta, por lo que cuando se genera la documentacion, no obtenemos los resultados que se esperarian.

**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)**

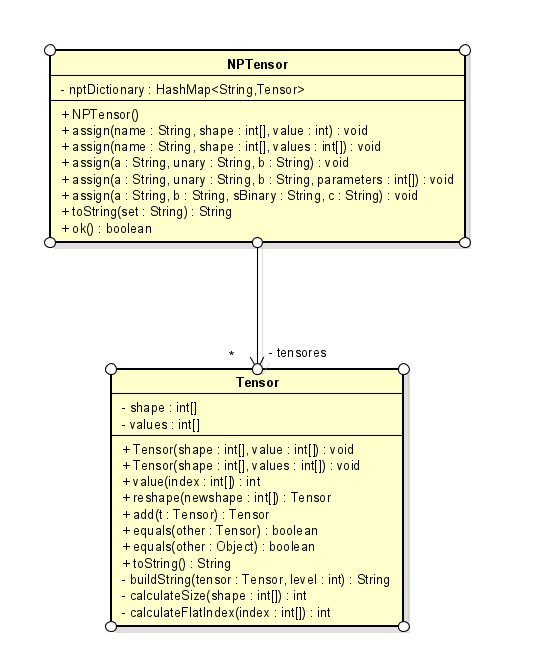
* **Tensor:** Contiene los metodos necesarios para ejecutar el archivo y compile, pero su codigo en muchos casos, tiene metodos “vacios” o sin codigo, ademas de errores como indentación o de documentación.
* **NPTensor:** Contiene los metodos necesarios para ejecutar el archivo y compile, pero su codigo en muchos casos, tiene metodos “vacios” o sin codigo, ademas de errores como indentación o de documentación.

**¿Qué son el código, la documentación y los comentarios?**

* El código, la documentación y los comentarios son todos importantes para la creación de programas informáticos. El código es lo que hace que un programa informático funcione, la documentación explica cómo usar el programa informático y los comentarios explican lo que hace el código.

**Ingenieria Reversa**

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



**2. ¿Cuál nuevo contenedor está definido? Consulte la especificación y el API Java 1¿Qué diferencias hay entre el nuevo contenedor, el ArrayList y el vector [] que conocemos?**

Los contenedores que se utilizan en el laboratorio son dos: HashMap y Array.

Las diferencias que hay entre ellos, es que HashMap almacena una pareja llave-valor, en una estructura de datos tipo HashTable, en donde además se permiten llaves nulas; el Array por otro lado, requiere saber su tamaño al momento de su inicialización, y almacena sus datos de manera secuencial en memoria

**Conociendo Pruebas en BlueJ**

Para poder cumplir con la 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 @)?**

Se encuentran las etiquetas @test, @BeforeClass y @before

Hay 6 metodos en total(beforeClass(), before(), shouldCreateTensor(), shouldKnowWhenTwoTensorAreEquals(), shouldRepresentATensorAsAString() y shouldAdd())

**¿cuantos métodos son de prueba? ¿cómo los reconocen?**

4 Metodos tiene la etiqueta @test, que son shouldCreateTensor(), shouldKnowWhenTwoTensorAreEquals(), shouldRepresentATensorAsAString() y shouldAdd(), y se reconocen por su etiqueta

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

* Se ejecuta unicamente la primera prueba, beforeClass() el cual arroja error, por lo que hasta no solucionar esto, no se van a ejecutar las demas y no pasa ninguna , pues los resultados de la prueba no coincide con lo que se espera.

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

@test es una etiqueta perteneciente al framework Junit, y es la encargada de decirle al compilador que el método ‘public void’ bajo ella puede ser ejecutado como un caso de prueba. Si el método no arroja ninguna excepción, Junit asumirá que el caso de prueba fue exitoso.

El codigo que se pone con @before es ejecutado antes de cada test, mientras que el codigo con @BeforeClass se corre una vez, antes de todas las otras pruebas del @test, por ejemplo si tenemos una clase con 10 tests, el codigo con etiqueta @Before se ejecutara 10 veces, pero el codigo con @BeforeClass se ejecutara una unica vez.

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

**assertTrue:** Toma como argumento una condición booleana, y se asegura de que esta sea verdadera, si no es así, arroja un AssertionError.

**assertFalse:** Toma como argumento una condición booleana, y se asegura de que esta sea falsa, si no es así, arroja un AssertionError.

**assertEquals:** su argumento son dos variables del mismo tipo, y esta función se encarga de verificar si esas dos variables tienen el mismo valor, para números de punto flotantes (float, double), se puede proporcionar un delta, es decir, el error máximo que puede ocurrir entre los dos valores.

**assertArrayEquals:** Se utiliza para verificar que dos arrays son iguales, y se asegura si es falsa o verdadera, si no es asi arroja un AssertionError.

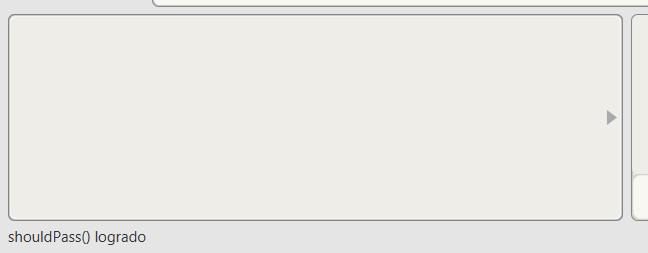
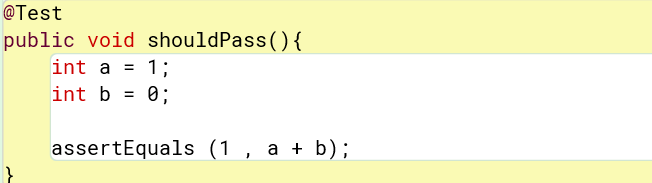
**assertNull:** Toma como argumento un objeto, cuyo valores debe ser null, si no es así, arroja un AssertionError.

**fail:** Falla un test, con un mensaje opcional.

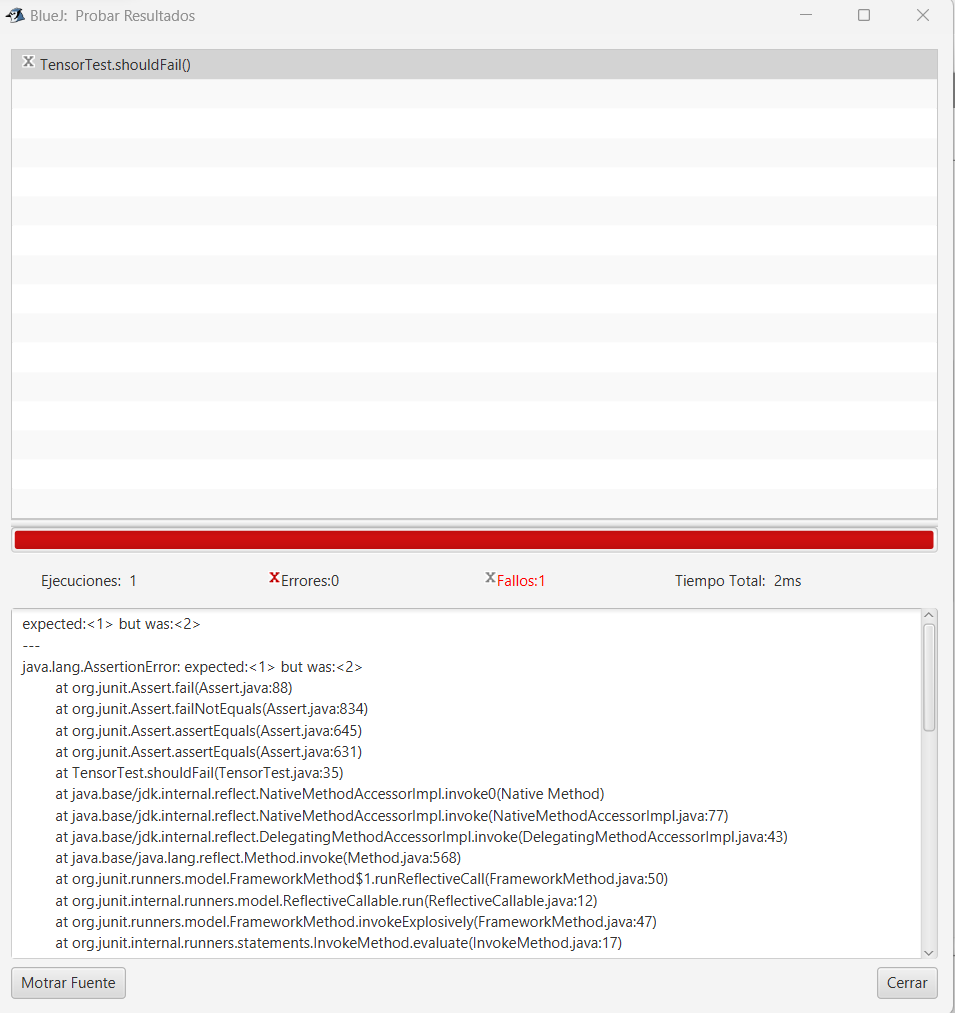
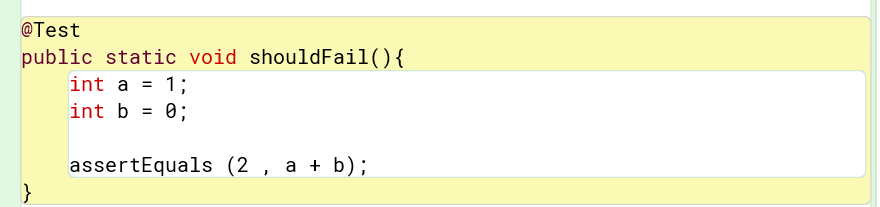
**5. Investiguen la diferencia que entre un fallo y un error en Junit. Escriba código, usando los métodos del punto 4., para lograr que los siguientes tres casos de prueba se comporten como lo prometen shouldPass, shouldFail, shouldErr.**

En JUnit, un fallo es una condición que se esperaba que se cumpliera, pero no lo hizo. Un error es una condición que no se esperaba que se produjera, pero sí lo hizo.

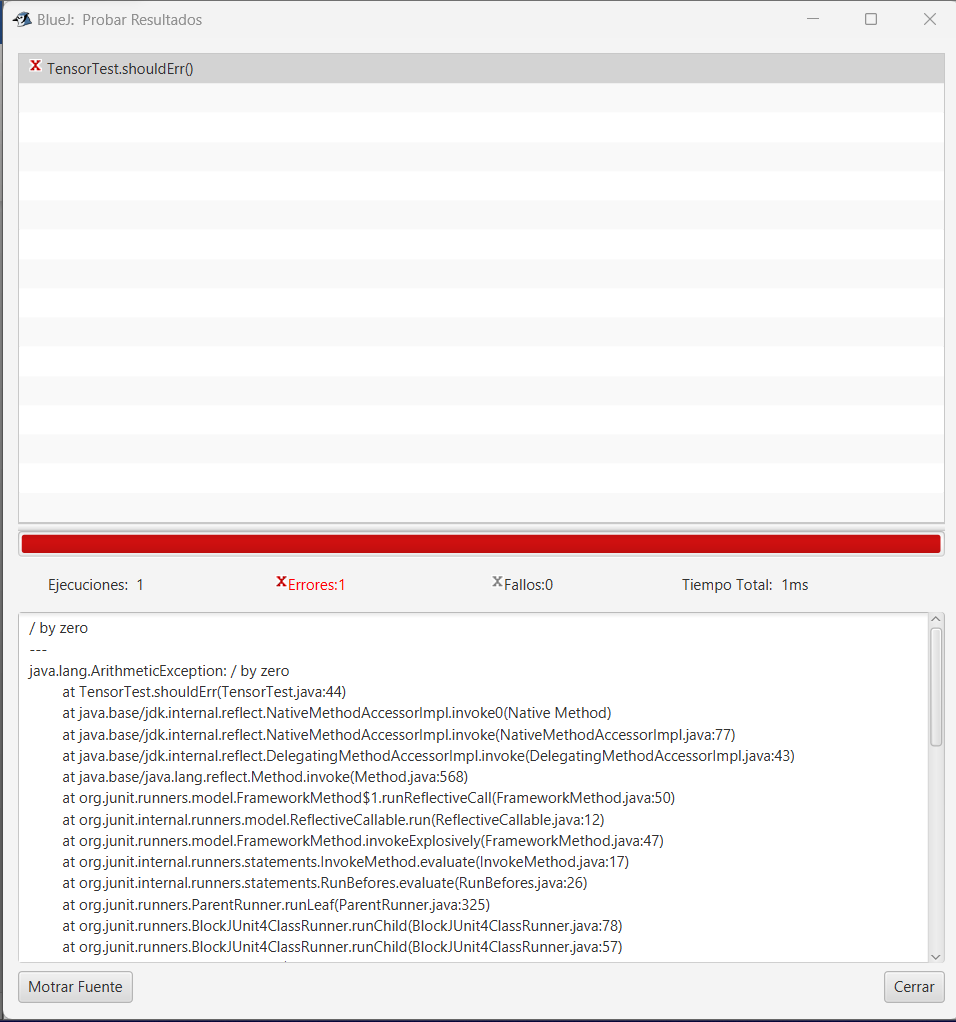
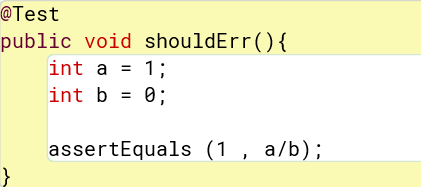
**ShouldPass()**



**ShouldFail()**



**ShouldErr()**



**Practicando Pruebas en BlueJ**

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

int [] shape un arreglo que contiene las dimensiones del arreglo que se le pase como parámetro.

int[] values que contiene los elementos que le ingresan al arreglo.

1. **Determinen el invariante de la clase Tensor. Justifique la decisión. B**

El invariante de la clase Tensor en Java es que el número de elementos en la matriz de valores es igual al producto de las dimensiones de la matriz de forma. Esto se garantiza en los constructores de la clase, que comprueban que la longitud de la matriz de valores sea igual al tamaño calculado a partir de la matriz de forma.

El invariante también se mantiene en los métodos de la clase, como reshape() y add(), que comprueban que las matrices de forma de los tensores sean iguales antes de realizar la operación.

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

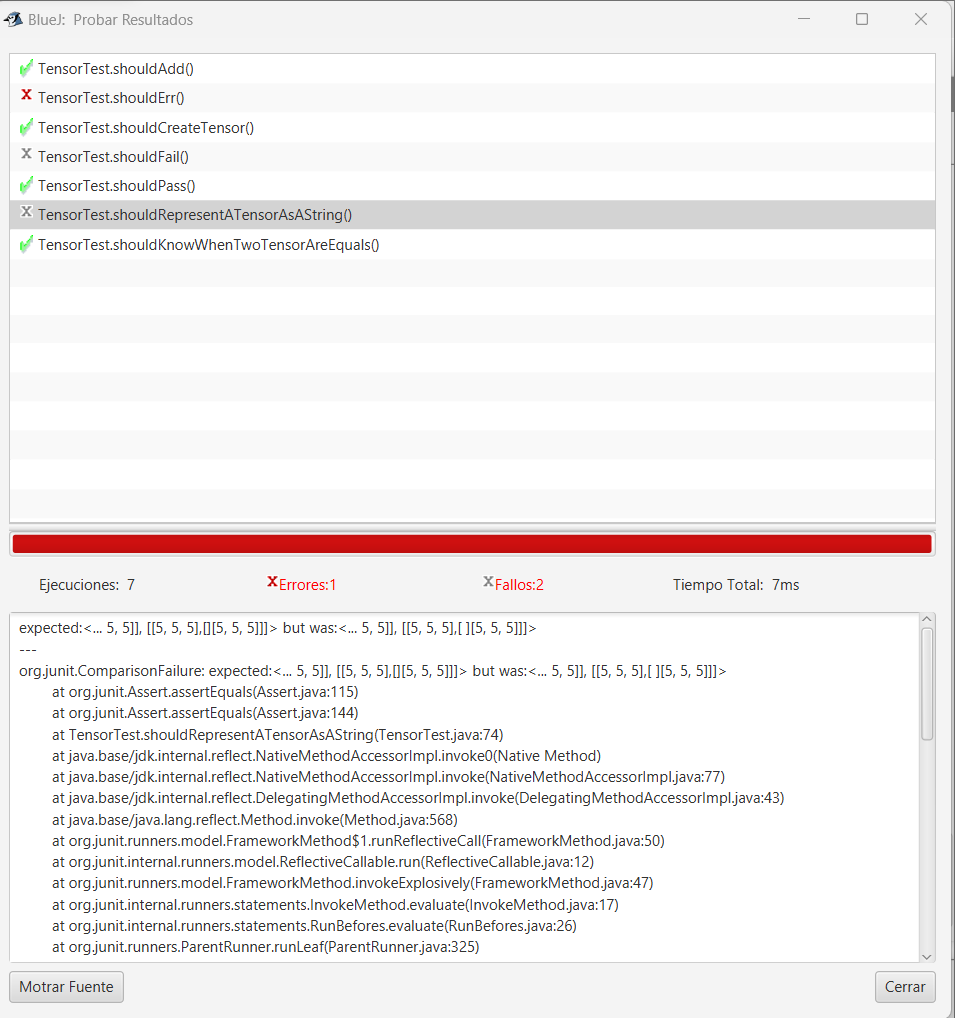
Se implementaron 2 constructores usando Override, en uno de ellos ingresan como parametros el arreglo de shape, y con todos los valores iguales establecidos con el valor dado. El otro constructor, lleva de igual forma el arreglo de shape, y otro arreglo con los valores de este.

Un metodo de acceso *(public int value(int[] index))* que devuleve el tensor en la posicion dada por el indice

2 metodos para transformar el tensor, *(public Tensor reshape(int[] newShape))* que nos devuelve el tensor con una nueva forma dada, *(public Tensor add(Tensor t))* devuelve la suma de este tensor y el tensor dado como parametro.

1 metodo de comparacion *(public boolean equals(Tensor other))* Devuelve un valor booleano de si el tensor es igual a otro dado como parametro.

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



Pasan las pruebas que deberian pasar y fallan las que deberian fallar, a excepción de shouldRepresentATensorAsaString(), la cual como podemos ver la entrada esperada y la obtenida son exactamente iguales, por lo que no se entiende porque falla.

**Desarrollando TensorNP**

BDD - MDD

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

**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 de 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**

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

Para este ciclo tenemos los metodos:

* + NPTensor()
  + assign(String name, int []) shape, int value)
  + assign(String name, int[]) shape, int[] values)
  + getValue(String name, int[] index)
  + ok()

2. Definir y programar los casos de prueba de esos métodos (piense en los deberia y los noDeberia (should and shouldNot)

**Declaración de Tensores:** Esta prueba evalúa la habilidad del usuario para declarar un tensor con una forma específica. Su importancia radica en la necesidad de crear el tensor con la forma correcta antes de asignar valores o realizar otras operaciones en él. Si esta función no opera adecuadamente, pueden surgir dificultades en la creación de tensores esenciales.

**Asignación de Valores:** En esta prueba, se verifica si el usuario puede asignar un valor específico a un tensor después de haberlo declarado. Esta capacidad es fundamental, ya que la asignación de valores a los elementos de un tensor es una operación común en el procesamiento de datos y en los cálculos numéricos.

**Consulta de Valores:** La prueba de consulta se lleva a cabo para asegurarse de que el usuario pueda recuperar de manera precisa los valores de un tensor utilizando su nombre y un índice. Este proceso es esencial para acceder y utilizar eficazmente los datos almacenados en los tensores.

3. Diseñar los métodos (Use diagramas de secuencia. En astah, adicione el diagrama al método)

* + En astah como NPTensor Casallas\_murcia/ 1. Ciclo 1

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

* En BlueJ en la clase de NPTensor en la seccion ciclo 1

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

* + 
  + 
  + 

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

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

* shape(Tensor t)
* reshape(Tensor t, int[] newShape)
* shuffle(String name)

2. Definir y programar los casos de prueba de esos métodos (piense en los deberia y los noDeberia (should and shouldNot)

* + **Caso de prueba 'testShapeNonExistent':** En este caso de prueba, se examina la función 'shape', la cual debe proporcionar la forma del tensor como una cadena. Se verifica si esta función retorna “null” y que no modifique las variables existentes.
  + **Caso de prueba 'testInvalidReshape':** En este caso, se evalúa cómo se gestiona la situación en la que se intenta remodelar un tensor con una forma incompatible respecto a la cantidad de elementos en el tensor. Se espera que esta acción provoque una excepción de tipo 'IllegalArgumentException'.
  + **Caso de prueba 'testShuffleNonExistent':** Este caso de prueba examina cómo se maneja la solicitud de mezcla de un tensor que no existe. Se espera que devuelva 'null' y que no tenga ningún impacto en las variables existentes.

3. Diseñar los métodos (Use diagramas de secuencia. En astah, adicione el diagrama al método)

* + En astah como NPTensor Casallas\_murcia/ 2. Ciclo 2

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

* En BlueJ en la clase de NPTensor en la seccion ciclo 2

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

* + 
  + 
  + 

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

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

* + slice (String name, int axis, int start, int end)
  + mean (String name, int axis)
  + find (String name, int value)

2. Definir y programar los casos de prueba de esos métodos (piense en los deberia y los no Deberia (should and shouldNot)

• **Caso de prueba 'testSlice':** En esta evaluación se comprueba la funcionalidad de la operación 'slice' para obtener una porción de un tensor.

• **Caso de Prueba 'testSliceInvalid':** En este caso de prueba, se examina la respuesta cuando se suministra un eje (axis) inválido a la operación 'slice'. Se verifica si la operación 'slice' devuelve 'null'.

• **Caso de Prueba 'testMean':** En esta instancia de prueba, se evalúa la funcionalidad de la operación 'mean' para calcular la media a lo largo de un eje de un tensor.

• **Caso de Prueba 'testFind':** En este caso de prueba, se verifica la funcionalidad de la operación 'find4' para localizar elementos específicos en un tensor.

3. Diseñar los métodos (Use diagramas de secuencia. En astah, adicione el diagrama al método)

* + En astah como NPTensor Casallas\_murcia/ 3. Ciclo 3

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

* En BlueJ en la clase de NPTensor en la seccion ciclo 3

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

* + 
  + 
  + 
  + 

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

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

* + suma (Tensor t1, Tensor t2)
  + resta (Tensor t1, Tensor t2)
  + multiplicacion(Tensor t1, Tensor t2)

2. Definir y programar los casos de prueba de esos métodos (piense en los deberia y los noDeberia (should and shouldNot)

* + **Caso de prueba ‘testSum’:** Se comprueba la funcionalidad del metodo de suma de tensores para verificar que la suma da correctamente.
  + **Caso de prueba ‘testRest’:** Se comprueba la funcionalidad del metodo de resta de tensores para verificar que la suma da correctamente.
  + **Caso de prueba ‘testMulti’:** Se comprueba la funcionalidad del metodo de multiplicacion de tensores para verificar que la multiplicacion da correctamente.

3. Diseñar los métodos (Use diagramas de secuencia. En astah, adicione el diagrama al método)

* + En astah como NPTensor Casallas\_murcia/ 3. Ciclo 3

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

* En BlueJ en la clase de NPTensor en la seccion ciclo 3

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

* + 
  + 
  + 

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

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

* + division (Tensor tensor1, Tensor tensor2)
  + cubeRoot(Tensor tensor1)
  + inverse (Tensor tensor)

2. Definir y programar los casos de prueba de esos métodos (piense en los deberia y los noDeberia (should and shouldNot)

* + **Caso de prueba ‘testDiv’:** Se comprueba la funcionalidad del metodo de division de tensores para verificar que la division da correctamente.
  + **Caso de prueba ‘testCubeRoot’:** Se comprueba la funcionalidad del metodo de raiz cubica de tensores para verificar que se de correctamente.
  + **Caso de prueba ‘testInverse’:** Se comprueba la funcionalidad del metodo de inverso de tensores para verificar que la inversa de correctamente.

3. Diseñar los métodos (Use diagramas de secuencia. En astah, adicione el diagrama al método)

* + En astah como NPTensor Casallas\_murcia/ 5. Ciclo 5

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

* En BlueJ en la clase de NPTensor en la seccion ciclo 3

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

Completen la siguiente tabla indicando el número de ciclo y los métodos asociados de cada clase

|  |  |  |
| --- | --- | --- |
| **Mini Ciclo** | **TensorNP** | **TensorNPTest** |
| 1 | * NPTensor() * assign (String name, int []) shape, int value) * assign (String name, int[]) shape, int[] values) * getValue(String name, int[] index) * ok (( | testAssignWithValue()  testAssignWithInvalidValues()  testGetValueWithInvalidName() |
| 2 | shape(Tensor t)  reshape(Tensor t, int[] newShape)  shuffle(String name) | testShapeNonExistent()  testReshapeInvalid()  testShuffleNonExistent() |
| 3 | slice(String name, int axis, int start, int end)  mean(String name, int axis)  find(String name, int value) | testSlice()  testSliceInvalid()  testMeanInvalid()  testFind() |
| 4 | suma(Tensor t1, Tensor t2)  resta(Tensor t1, Tensor t2)  multiplicacion(Tensor t1, Tensor t2) | testSum()  testRest()  testMulti() |
| 5 | division(Tensor t1, Tensor t2)  cubeRoot(Tensor t1)  inverse(Tensor t1) | testDiv()  testCubeRoot()  testInverse() |

**RETROSPECTIVA**

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

* + **25 horas por persona**

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

* + Esta completo, pero faltan pulir algunos detalles.

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

* + **La práctica más útil fue la de unit testing, ya que poniéndola en práctica se pudieron hacer pruebas relevantes que mejoraran el proceso de desarrollo.**

4. ¿Cuál consideran fue el mayor logro? ¿Por qué?

* + **Aprender el proceso de desarrollo orientado a pruebas, a medida que se iba avanzando en el laboratorio, se notó una mejoría en la habilidad para plantear casos de prueba relevantes, y un modelamiento adecuado.**

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

* + **Aprender a definir pruebas relevantes y modelar la situación antes de codificar la solución, este proceso requirió tiempo de planeación de cómo iba a ser la ejecución del programa y los posibles casos en los que podía salir mal.**

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

* + La programación a pares que se vio en las practicas XP, ya que fue clave para poder culminar el laboratorio