**Laboratorio 2**

**Santiago Guladrón- Laura Sofia Gil**

**Conociendo el proyecto [En lab02.doc]**

1. El proyecto “dataMatrixCalculator” 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.

El archivo *“dataMatrixCalculator”,* Están 2 archivos importantes: un astah donde se muestra la estructura grafica del proyecto con respecto a las clases; y un archivo donde está todo el código del proyecto, además de un archivo doc. donde están toda la información de los documentos HTML con js y css.

Las extensiones de cada uno de los archivos son:

* .html : Hace referencia a los archivos ejecutados que resumen los contenidos y objetivos de cada uno de los otros archivos al momento de la ejecución.
* .js : extensión para la interacción de las páginas HTML que están en el archivo doc.
* .css : la codificación para mostrar el diseño estándar de la documentación de los archivos HTML.
* .java: Archivo que está escrito con la codificación de Java y está listo para ser ejecutado por cualquier programa que pueda leerlo y entenderlo.
* .class En Java, un archivo ".class" es un archivo binario que contiene el bytecode compilado de un programa Java. El bytecode es un código intermedio que se ejecuta en la Máquina Virtual de Java (JVM).
* .bluej Los proyectos en BlueJ, como los paquetes estándar en Java, son directorios conteniendo los archivos incluidos en el proyecto
* .ctxt Contiene la documentación de la clase
* .txt El directorio README que se forma por esta extensión es una visualización de la descripción del proyecto.
* .asta es el archivo de tipo UML que permite la creación y en este caso visualización de los diagramas para visualizar y documentar el diseño de software

2. Exploren el proyecto en BlueJ

¿Cuántas clases tiene?

Tenemos 3 clases

¿Cuál es la relación entre ellas?

La relación entre las clases;

DataMatrix y Data CalculatorMatrix se relacionan mediante que Data matriz es la base para realizar el DataCalculatorMatrix

¿Cuál es la clase principal de la aplicación?

La clase principal de la aplicación es DataMatrix

¿Cómo la reconocen?

Se reconocen por medio de que DataMatrix en una superclase ya que las otras dos clases se dirigen hacia ella por medio de las relaciones y métodos

¿Cuáles son las clases “diferentes”?

La clase diferente es: Data Test

¿Cuál es su propósito?

Su propósito es: Lo que los diferencia es que Data Test nos da un caso de prueba de unidad de Data que evalúan las acciones de la clase de Data.

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?

No, no esta completa la documentación de cada clase, porque en la documentación inicial de algunas clases no tiene el detalle de para que es o el autor o versión.

(Detallen el estado de documentación de cada clase: encabezado y métodos)

La clase DataMatrix

-Encabezado incompleto: public class DataMtrix

-Metodos incompletos :

public DataMatrix(String [][] data)

public DataMatrix shape()

public String toString(int row, int column)

public DataMatrix reshape(int row, int column)

public DataMatrix add(DataMatrix t)

public boolean equals(DataMatrix other)

public boolean equals(Object other)

public String toString ()

La clase Data

-Encabezado completo: public class Data

-métodos incompletos:

public String toString ()

La clase DataMatrixCalculator

-Encabezado incompleto: public class DataMatrixCalculator

-Metodos incompleto:

public DataMatrixCalculator()

public void assign(String name, String values[][] )

public String[] variables()

public void assign(String a, char unary, String b)

public void assign(String a, String b, char sBinary, String c)

public String toString(String variable)

public boolean ok()

- 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 DataMatrixCalculator:**

**Atributos y métodos:**

• Atributos: tiene un atributo privado declarado como un Hashmap y llamado variables, a pesar de que se forma este, no hay ningún otro que haga referencia a la definición de DataMatrix.

• Métodos: cuenta con 6 metodos que refieren a assign, variables, toString y ok, unas respectivamente hacen asignación genérica dependiendo los parámetros, otro como representación de variables, el toString la representación de las variables y el ok verificar si los métodos son correctos. Ninguno de estos métodos tiene una documentación clara para poder precisar el uso que debería tener para el entendimiento y próxima construcción del código.

**Código, Documentación y Comentarios:**

• Código: El código de la clase DataMatrixCalculator está implementado correctamente y sigue las convenciones de estilo Java. Los atributos y métodos se declaran correctamente y no se observan errores de sintaxis en el código.

• Documentación: Esta clase contiene comentarios de documentación básica al inicio del archivo, que proporcionan información sobre el nombre del archivo y el autor. No obstante, carece de documentación completa de atributos y métodos dentro de las clases.

• Comentarios en el código: La clase cuenta con pocos comentarios en el código que describen la funcionalidad de los métodos, lo cual resulta beneficioso para entender su funcionalidad. No obstante, estos comentarios son limitados y no ofrecen una descripción pormenorizada de los parámetros, las salidas esperadas ni los detalles de la implementación.

**Clase DataMatrix**

Atributos y Métodos:

• Atributos: La clase DataMatrix presenta un atributo privado que es tipo data siendo un arreglo bidimensional.

• Métodos: La clase DataMatrix incluye diversos métodos como el método shape devuelve una nueva instancia que representa las dimensiones, toString(int row, int column) retorna la cadena del elemento en los índices dados, reshape(int row, int column) crea un nuevo DataMatrix con las dimensiones indicadas, add(DataMatrix t) combina matrices devolviendo la suma de elementos,El método equals(DataMatrix other) compara matrices y devuelve un resultado booleano, El método equals(Object other) convierte y llama al equals especializado, toString() devuelve una representación en cadena de la matriz. Actualmente, retorna null y debería implementarse para proporcionar una representación significativa.

**Código, Documentación y Comentarios:**

* Código: El código de la clase DataMatrix está implementado correctamente y sigue las convenciones de estilo Java. Los atributos y métodos se declaran correctamente y no se observan errores de sintaxis en el código.

• Documentación: Esta clase contiene comentarios de documentación básica al inicio del archivo, que proporcionan información sobre el nombre del archivo y el autor. No obstante, carece de documentación completa de atributos y métodos dentro de las clases.

• Comentarios en el código: La clase cuenta con pocos comentarios en el código que describen la funcionalidad de los métodos, lo cual resulta beneficioso para entender su funcionalidad. No obstante, estos comentarios son limitados y no ofrecen una descripción pormenorizada de los parámetros, las salidas esperadas ni los detalles de la implementación.

**Clase Data**

Atributos y Métodos:

• Atributos: La clase Data no presenta ningún atributo.

• Métodos: Los métodos add y sub realizan operaciones de suma y resta, respectivamente, considerando la lógica específica para cada tipo de dato. El método type devuelve el tipo de dato ('b', 'n' o 'c'). El método toString proporciona una representación en cadena del objeto, mientras que string devuelve la cadena utilizada para crearlo, eliminando espacios en blanco.

**Código, Documentación y Comentarios:**

* Código: El código de la clase Data está implementado correctamente y sigue las convenciones de estilo Java. Los atributos y métodos se declaran correctamente y no se observan errores de sintaxis en el código.

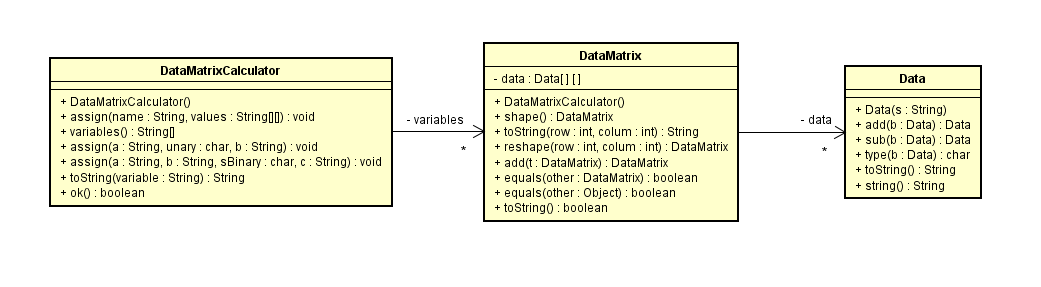
• Documentación: Esta clase contiene comentarios de documentación básica al inicio del archivo, que proporcionan información sobre el nombre del archivo y el autor. No obstante, carece de documentación completa de atributos y métodos dentro de las clases.

• Comentarios en el código: La clase cuenta con pocos comentarios en el código que describen la funcionalidad de los métodos, lo cual resulta beneficioso para entender su funcionalidad. No obstante, estos comentarios son limitados y no ofrecen una descripción pormenorizada de los parámetros, las salidas esperadas ni los detalles de la implementación.

**Ingeniería reversa [En lab02.doc DataMatrixCalculator.asta]**

**MDD MODEL DRIVEN DEVELOPMENT**

1. Complete el diagrama de clases correspondiente al proyecto



2. ¿Cuáles contenedores están definido?

El nuevo contenedor que esta definido es **HashMap** que admite una concurrencia total de recuperaciones y una alta concurrencia esperada para las actualizaciones. A pesar de que no admite duplicados su base en la búsqueda de elementos se basa en las claves que le son dadas a cada uno de sus elementos, es decir se mantiene una regla y relación entre clave y el valor que representa.

¿Qué diferencias hay entre el nuevo contenedor, el ArrayList y el vector [] que conocemos?

Consulte el API de java. ¿Cómo adicionamos un

elemento?

1. Hashmap no acepta duplicados, mientras que ArrayList y Vector si lo hacen.

2. El orden de inserción tanto en vector como en ArrayList es muy similar, sin embargo esto causa problemas al momento de eliminar y/o buscar un elemento en especifico, ya que en ocasiones es necesario que los elementos modifiquen su posición para mantener la estructura.

3. La representación de los índices de los elementos se reconocen por ser unas claves en un HashMap y por ser índices con un orden en un ArrayList y un vector.

La principal distinción entre HashMap, ArrayList y Vector radica en que Vector es una implementación sincronizada, lo que significa que solo un único subproceso puede acceder a un método de Vector a la vez. En cambio, ArrayList no es una implementación sincronizada, lo que permite que múltiples subprocesos accedan simultáneamente a un ArrayList, lo que lo hace adecuado para entornos multiproceso. Además, en HashMap, los elementos se identifican mediante claves definidas como "item1", "item2", "persona1", "ciudad2", etc. Esta relación se mantiene constante y constituye su principal ventaja.

¿Cómo lo consultamos? ¿Cómo lo eliminamos?

Para trabajar con estructuras de datos en Java, como ArrayList, Vector y contenedor se puede agregar elementos utilizando el método add() para ArrayList y Vector, y put() para el contenedor. Para consultar, emplea get() especificando el índice en ArrayList y Vector, y la clave en el contenedor. Para eliminar, utiliza remove() proporcionando el índice en ArrayList y Vector, y la clave en el contenedor.

**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 implementaremos algunos métodos en la clase DataTest

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

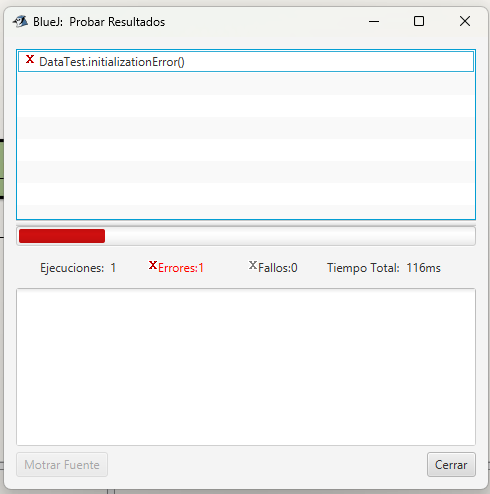
Tiene 7 etiquetas: 5 de tipo Test , 1 de tipo before y otro de tipo BeforeClass

Tiene 7 métodos:

* beforeClass()
* before()
* shouldCreateData()
* shouldRepresentADataAsAString()
* shouldKnowWhenTwoBooleansAreEquals ()
* shouldKnowWhenTwoNumbersAreEquals()
* shouldKnowWhenTwoCharactersAreEquals ()

Tiene 5 métodos de prueba reconocidos por @test reconocidos por jUnit como métodos de prueba y se ejecutan al momento de hacer uso de las pruebas.

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



Se ejecuta solo una prueba, donde se ejecuta un error directamente y ninguna pasa por un problema de inicialización.

3. Estudie las etiquetas encontradas en 1 (marcadas con @). Expliquen en sus palabras su significado.

* ***@BeforeClass***: Esta anotación es un indicador que el método debe ser ejecutado antes de realizar toda prueba dentro de la clase.
* ***@Before***: Esta anotación es un indicador que el método debe ser ejecutado antes de realizar una prueba especifica dentro de la clase.
* ***@Test:*** Esta anotación es un indicador que el método hace parte de las pruebas unitarias y es reconocido por ser un método de prueba.

4. Estudie los métodos assertTrue, assertFalse, assertEquals, assertArrayEquals, assertNull y fail de la clase Assert del API JUnit 1 . Explique en sus palabras que hace cada uno de ellos.

* ***Fail(String message):*** Utilizado para forzar una falla en la prueba y proporcionar un mensaje de error personalizado. Se emplea cuando es necesario indicar explícitamente que la prueba debe fallar en una parte específica del código.
* ***AssertArrayEquals(expectedArray, actualArray***): Este método se utiliza especialmente para comparar matrices y calcula cada elemento de la matriz esperada (expectedArray) con la matriz real (actualArray). Si el contenido y la longitud de las dos matrices son iguales, la verificación se considera exitosa; de lo contrario, se registra un error.
* ***AssertEquals(expected, actual):*** Método que compara el valor esperado con el valor real. Si ambos valores son iguales, la prueba ha pasado. Si difieren, la prueba se marca como fallida. Este método se utiliza para comprobar la similitud entre el resultado y su valor esperado.
* ***AssertTrue(boolean condition***): Método que se utiliza para verificar si la condición dada es verdadera. Cuando la condición es verdadera, la prueba continúa sin problemas; de lo contrario, la prueba falla y se detiene. Este método se utiliza para garantizar que la expresión siga siendo verdadera durante la ejecución de la prueba.
* ***AssertFalse(boolean condition)***: Este método verifica si la condición es falsa. Si la condición es falsa, la prueba continúa; de lo contrario, se considera un fracaso.
* ***AssertNull(object):*** Este método comprueba si el objeto dado es nulo. Si el objeto es nulo, la prueba pasa; si el objeto no es nulo, la prueba se considera fallida.

5. Investiguen y expliquen 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 (failure) señala que una expectativa particular no se ha cumplido, pero la ejecución de la prueba prosigue para evaluar otros partes del código.

Por otro lado, un error (error) indica que la propia prueba está defectuosa o no puede ejecutarse debido a problemas del cuerpo de las pruebas o excepciones no gestionadas en el código de prueba.

import org.junit.Assert;

import org.junit.Test;

public class TestCases {

@Test

public void shouldPass() {

// Caso que debería pasar sin problemas

int result = 2 + 2;

Assert.assertEquals("La suma no es la esperada", 4, result);

Assert.assertTrue("La condición no es verdadera", result > 0);

Assert.assertFalse("La condición no es falsa", result < 0);

Assert.assertNull("El objeto no es nulo", null);

Assert.fail("Este mensaje nunca se mostrará porque la prueba pasa correctamente");

}

@Test

public void shouldFail() {

// Caso que debería fallar debido a una aserción incorrecta

int result = 2 + 2;

Assert.assertEquals("La suma no es la esperada", 5, result);

}

@Test

public void shouldErr() {

// Caso que debería generar un error debido a una excepción no controlada

int result = 2 / 0; // División por cero, generará una ArithmeticException

Assert.assertEquals("La división no es válida", 4, result); // Esta línea nunca se alcanzará debido al error anterior

}

}

**Prácticando Pruebas en BlueJ [En lab02.doc \*.java]**

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

1. Determinen los atributos de la clase Data. Justifique la selección.

Para los atributos de la clase Data, se debe considerar el tipo de dato que es (‘b’, ‘n’, ‘c’), además de almacenar la variable en cadena como entro; por ultimo agregar 3 módulos como “nulos” donde cada uno tiene los tipos: boolean, character, integer.

(moduloc, modulob, moduloi).

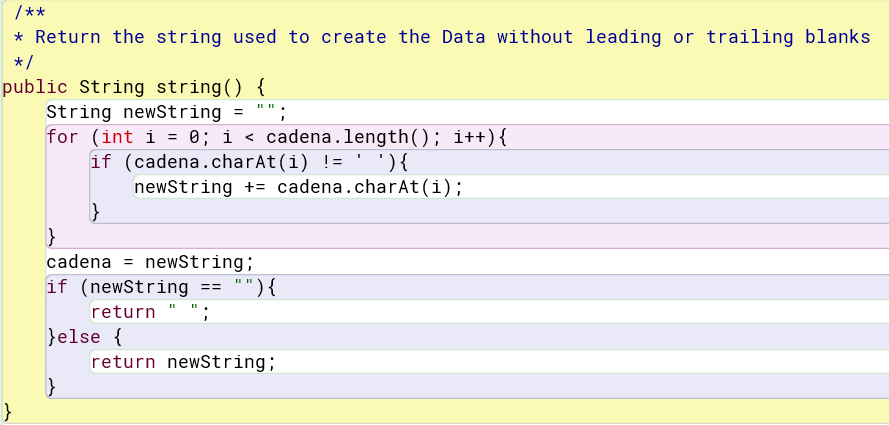
1. Determinen el invariante de la clase Data. Justifique la decisión.

Para la invariante, consideramos que lo mas importante que no cambia durante la manipulación del Data, seria Type, ya que este nos indica el tipo de dato que es el Data, y a partir de esto podemos hacer las sumas y restas para DataMatrix.

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

Para poder pasar las pruebas de unidad definidas se necesitan crear 3 métodos:

* string():



* toString():

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamenteInterfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

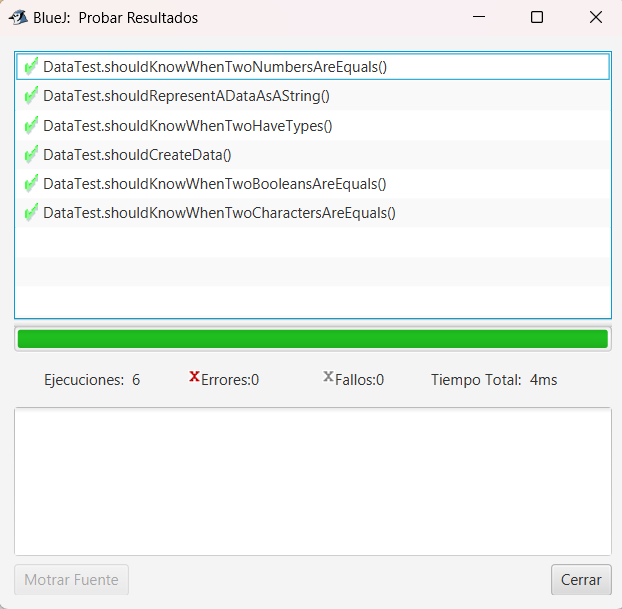
Descripción generada automáticamente

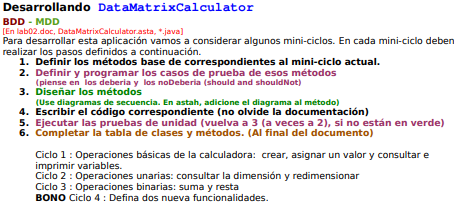
* equals():

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

1. Capturen los resultados de las pruebas de unidad.





Ciclo1:

**Constructor DataMatrixCalculator ():** Crea una instancia de la clase DataMatrixCalculator inicializando un nuevo objeto DataMatrixCalculator con una colección de variables(HashMap) y estableciendo el indicador en false.

**Método assign(String name, String values[][] ):** Inserta en el HashMap “variables” un String con el nombre de la matriz mas los valores que tiene en sus filas y columnas.

**Método variables():** Le muestra al usuario todos los nombres de las matrices que están dentro de el HashMap “variables”.

**Método assign(String a, char unary, String b):** Se guarda en la variable a, la matriz que hará la operación “unary” (puede ser r(transpuesta) o s(forma)) sobre la matriz que esta guardada en “variables” por el nombre dado en b

**Método assign(String a, String b, char binary, String c):** Se guarda en la variable a, la matriz del resultado al hacer la operación “binary” (puede ser “+”(suma) o “-”(resta) de matrices) sobre las matrices guardadas en “variables” que tienen los nombres dados en b y c

**Método toString(String variable):** Retorna la representación de la matriz guardada en “variables” con el nombre dado en “variable”, donde es una cadena con los valores consecutivos separados por espacios.

**Método ok():** Ofrece un valor booleano que señala si la última operación resultó exitosa. Inicializado como "false" en el constructor, este valor puede ser alterado en función de las operaciones ejecutadas. Su utilidad radica en la capacidad para comprobar si se produjeron errores en las operaciones previas.

Ciclo 2:

**Método getUnaryDimension(String name, String unary):** Nos muestra las dimensiones que tiene una matriz guardada en “variables” con nombre “name” y depende de la función que le pidamos en “unary”(“s” para las dimensiones normales “s” para las dimensiones invertidas).

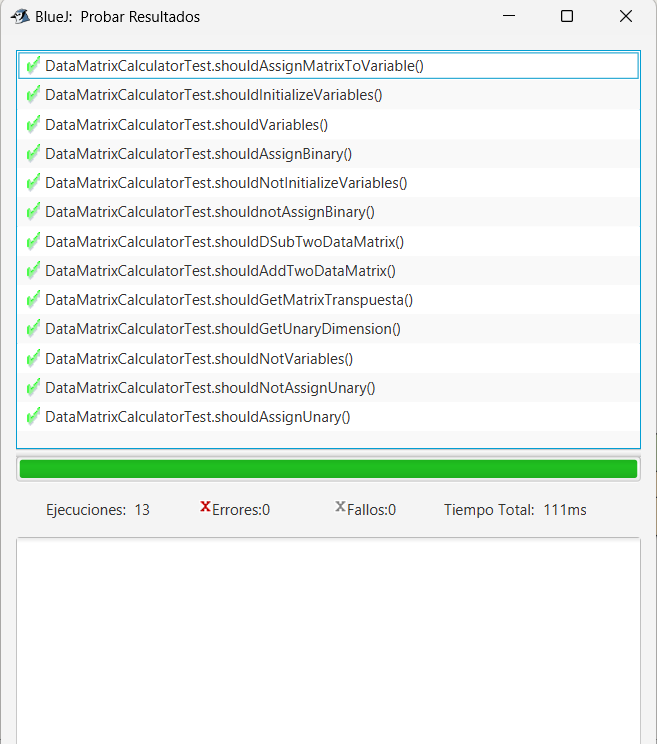
**Método transpuesta(String name):** Muestra cual es la matriz transpuesta dada por el nombre “name” que esta guardada en “variables”.

Ciclo 3:

**Método suma(String nameResult, String nameA, String nameB):** Guarda en el HashMap “variables” una nueva matriz de nombre “nameResult” donde esta la suma entre la matriz de nombre “nameA” con “nameB”.

**Método resta(String nameResult, String nameA, String nameB):** Guarda en el HashMap “variables” una nueva matriz de nombre “nameResult” donde está la resta entre la matriz de nombre “nameA” con “nameB”.

Casos de prueba para todos los Ciclos 1, 2 y 3:



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

|  |  |  |
| --- | --- | --- |
| Mini-Ciclo | DataMatrixCalculator | DataMatrixCalculatorTest |
| 1 | DataMatrixCalculator() | shouldInitializeVariables()  shouldNotInitializeVariables() |
| 1 | Assign(String name, String values[][]) | shouldAssignMatrixToVariable() |
| 1 | variables() | shouldVariables()  shouldNotVariables() |
| 1 | assign(String a, char unary, String b) | shouldAssignUnary()  shouldNotAssignUnary() |
| 1 | assign(String a, String b, char binary, String c) | shouldAssignBinary()  shouldnotAssignBinary() |
| 1 | toString(String variable) |  |
| 1 | ok() |  |
| 2 | getUnaryDimension(String name, String unary) | shouldGetUnaryDimension() |
| 2 | transpuesta(String name) | shouldGetMatrixTranspuesta() |
| 3 | suma(String nameResult, String nameA, String nameB) | shouldAddTwoDataMatrix() |
| 3 | resta(String nameResult, String nameA, String nameB) | shouldDSubTwoDataMatrix() |

**RETROSPECTIVA**

1. ¿Cuál fue el tiempo total invertido en el laboratorio por cada uno de ustedes?

(Horas/Hombre)

Este laboratorio tuvo un tiempo total de aproximadamente 18 horas (9 por persona)

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

El laboratorio quedo completo a excepción del bono, por temas de organización del tiempo

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

La practica XP mas útil para nosotros es el trabajar en dúos donde cada uno está al pendiente de que es lo que hace el otro, para saber en que se equivoco y que ideas tiene por desarrollar

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

Lograr darle un funcionamiento completo a las sumas, restas y demás métodos para la interacción entre matrices por “unary” o “binary”.

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

Completar todos los casos Test, puesto que había algunas funciones que estaban bien definidas pero no sabíamos cómo desarrollar los ejemplos para las pruebas.

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

resultados?

Tener una comunicación clara respecto a lo que adelantamos y lo que vamos a hacer a continuación en cualquiera de los archivos del laboratorio, nos comprometemos a optimizar mejor los tiempos para tener tiempo de sobra al completar el laboratorio.