**Laboratorio 02**

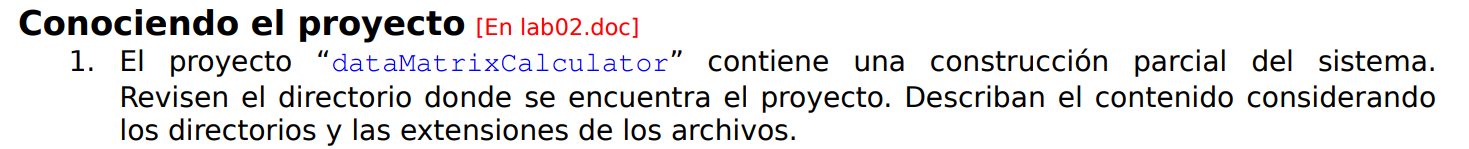
**Integrantes:**

**Sebastián Cardona Parra**

**Diego Alexander Cárdenas Beltrán**

**Universidad Escuela Colombiana de Ingeniería Julio Garavito**  
**Programación Orientada a Objetos**  
 **María Irma Rozo**

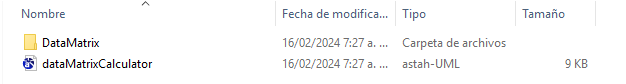
**16/02/2024**



Primer directorio:

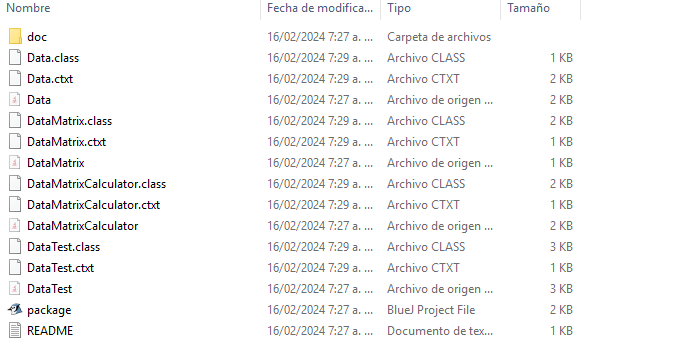


Entrando a dicho directorio:



Se ve que tenemos un directorio llamado DataMatrix y un archivo de extensión .astah dentro del directorio principal 2024-01-dataMatrixCalculator

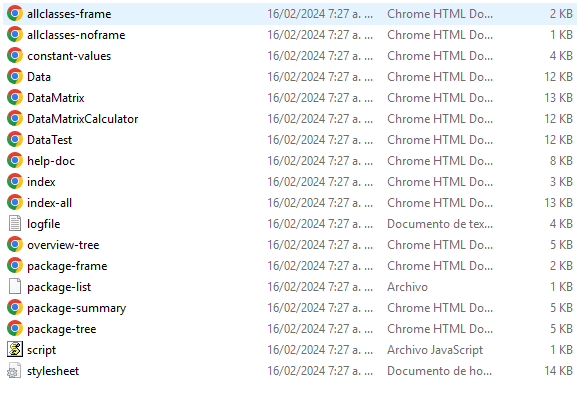
Entrando al directorio DataMatrix:



Se ve un nuevo directorio llamado doc y varios archivos de extensiones .java, .class, .ctxt, .txt

Estos archivos contienen toda la codificación y el proyecto como tal de DataMatrixCalculator

Dentro del directorio doc se encuentra:



Hay varios archivos .HTML que contienen la documentación del proyecto y otros archivos .txt, .JavaScript y .css





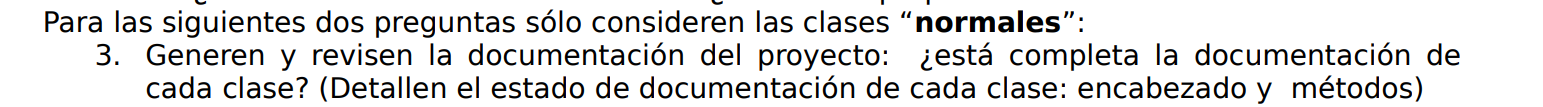
Tiene 4 clases y la relación entre ellas es:

* DataMatrixCalculator esta relacionado con la clase DataMatrix,
* DataTest esta relacionado con Data pero DataTest es una clase de testeo.



DataMatrixCalculator, debido al contexto de los códigos pudimos analizar que DataMatrixCalculator es la principal debido a que trae los datos de DataMatrix y DataMAtrix de Data de una manera relacionada, además el objetivo del proyecto es hacer cálculos con matrices, eso lo permite la clase DataMatrixCalculator 

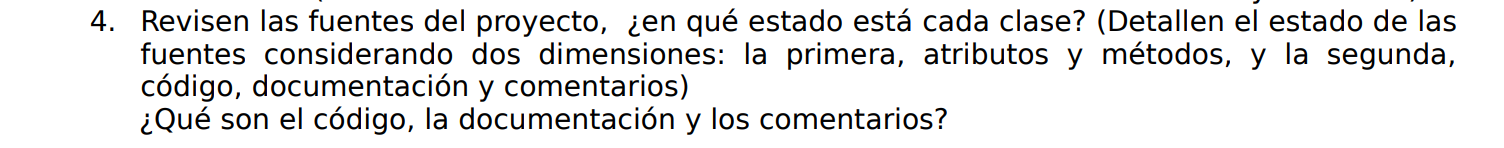
La clase diferente es DataTest, su propósito como su nombre lo dice es testear los métodos de la clase Data y verificar que estén bien implementados



DataMatrixCalculator: No tiene ningún tipo de documentación de JavaDoc, tiene unos comentarios que dan un tipo de contexto de lo que supuestamente se debe de realizar, y en ecabezado solo tiene el autor.

DataMatrix: Solo tiene la documentación de un constructor, y en encabezado solo tiene el autor y tiene dos @Override.

Data: Tiene documentación de todos sus métodos, y comentarios que aportan a su funcionalidad, y contiene el encabezado con su descripción.



**Clase DataMatrix:**

Atributos y métodos: Define atributos privados data y métodos públicos como shape(), toString(int row, int column), reshape(int row, int column), add(DataMatrix t), equals(DataMatrix other) y los métodos equals() y toString() sobrescritos.

Código, documentación y comentarios:

El código de los métodos está incompleto (toString() retorna null, por ejemplo).

Hay comentarios de documentación de los métodos, pero estos no explican completamente qué hace cada método ni cómo se espera que funcione.

**Clase DataMatrixCalculator:**

Atributos y métodos: Define un atributo variables y métodos públicos como assign(String name, String values[][] ), variables(), assign(String a, char unary, String b), assign(String a, String b, char sBinary, String c), toString(String variable) y ok().

Código, documentación y comentarios:

El código de los métodos está incompleto.

Hay comentarios de documentación de los métodos, pero no detallan completamente el funcionamiento esperado.

Se proporciona una breve descripción de la clase, pero no se detalla su funcionalidad en profundidad.

**Clase Data:**

Atributos y métodos: No hay atributos explícitos definidos, solo los métodos públicos add(Data b), sub(Data b), type(Data b), toString() y string().

Código, documentación y comentarios:

El código de los métodos está incompleto.

Hay comentarios de documentación de los métodos, pero no detallan completamente el funcionamiento esperado.

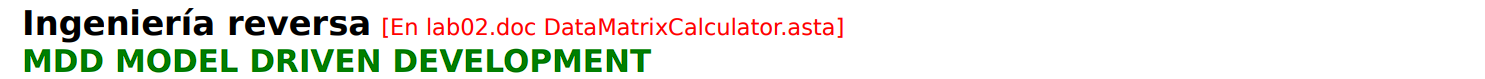
Se proporciona una breve descripción de la clase, pero no se detalla su funcionalidad en profundidad.

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

**Código:** Es la implementación real de los métodos y atributos, es decir, el conjunto de instrucciones que determina el comportamiento y la lógica de la clase.

**Documentación:** Es la información que describe el propósito, la funcionalidad y el uso de la clase, sus atributos y métodos. La documentación puede incluirse dentro del código, por ejemplo, utilizando Javadoc.

**Comentarios:** Son fragmentos de texto dentro del código que explican ciertas partes del mismo. Pueden proporcionar aclaraciones sobre decisiones de diseño, explicaciones sobre el funcionamiento de un método, advertencias sobre posibles problemas, etc. Los comentarios son útiles para que otros programadores comprendan el código más fácilmente.



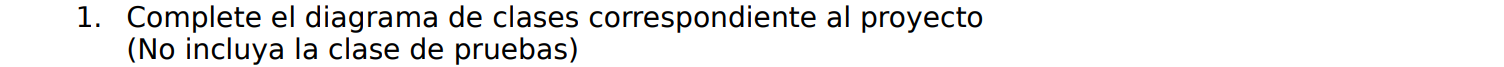
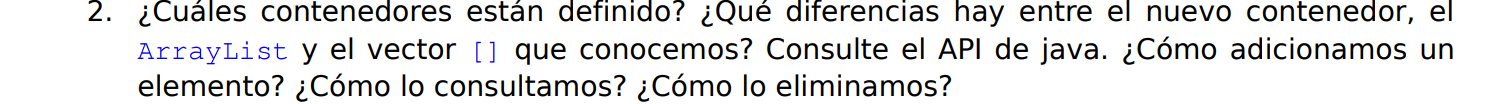


Diagrama hecho en astah

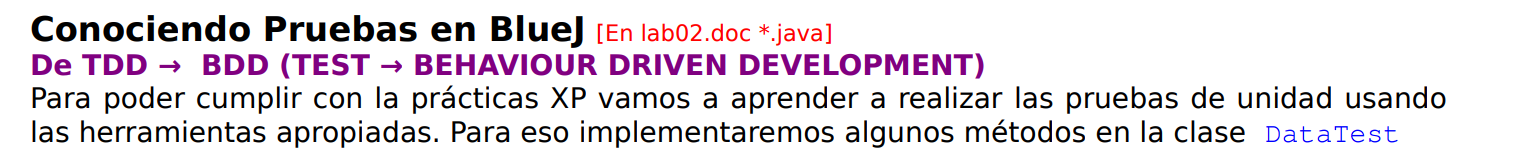


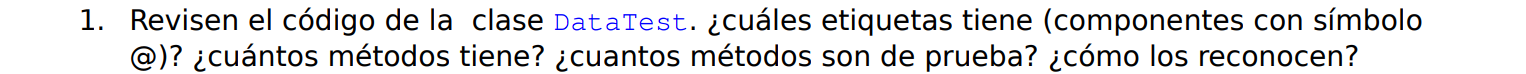
Está definido el contenedor de HashMap, y de vector, específicamente vector de vectores, el HashMap, almacena los datos tipo clave-valor, el ArrayList define una secuencia de datos dinámico, mientras el vector, la secuencia de datos es estático, en tanto rendimiento, el HashMap tiene ventaja en operaciones de insertar y eliminar pues su complejidad es de O(1) mientras en el vector y ArrayList es de O(n), el tamaño del vector es estático y se define en su declaración, mientras que en el HashMap y en un ArrayList, su tamaño es dinámico.

Para insertar un elemento en el HashMap se usa el método .put(clave,valor), en ArrayList se usa el .add() y en los vectores se usa el nombrevector[pos] = elemento, si quieres añadir otro, es decir, extender el tamaño del vector, no es posible, debes crear otro con el elemento a añadir

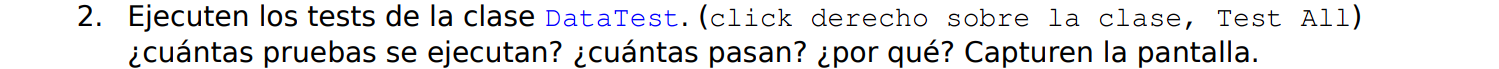
Para consultarlo en un HashMap se una el método .get(clave), en ArrayList es .get(index) y en un vector se usa: nombre[posicion]

Para eliminarlo en un HashMap se una el método .remove(clave), en el ArrayList es .remove(index) y en un vector, directamente no es posible, debes crear otro vector sin el elemento que deseas eliminar





Tiene 7 @, 5 son métodos de prueba, por la palabra test podemos asociarlos a las pruebas necesarias.



Se ejecuta una prueba, no pasa ninguna

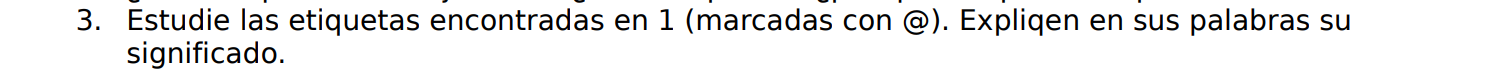
Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

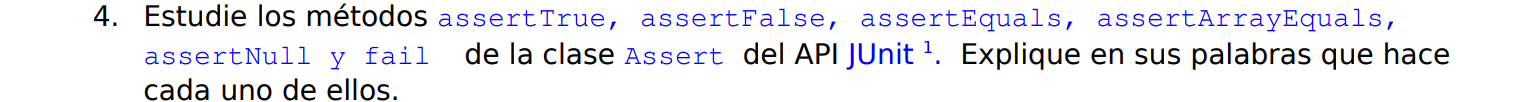
El método **beforeClass()** debe ser estático para que pueda ser invocado correctamente por JUnit.

Interfaz de usuario gráfica, Texto, Aplicación

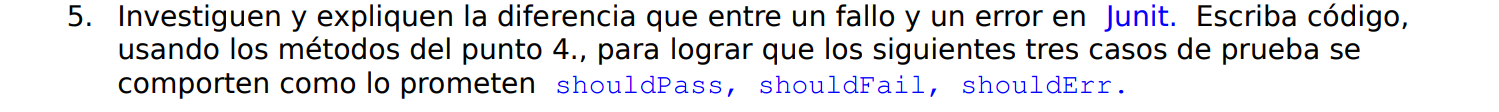
Descripción generada automáticamente



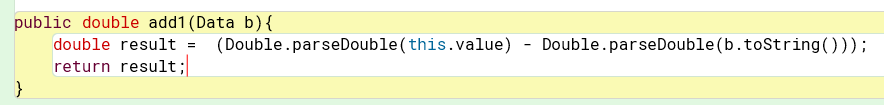
* **@BeforeClass:** Esta anotación se utiliza para indicar que un método específico debe ejecutarse una vez antes de que se ejecuten todos los métodos de prueba en la clase.
* **@Before:** Similar a @BeforeClass, pero indica que un método específico debe ejecutarse antes de cada método de prueba en la clase.
* **@Test:** Esta es la anotación principal que marca un método como un método de prueba.



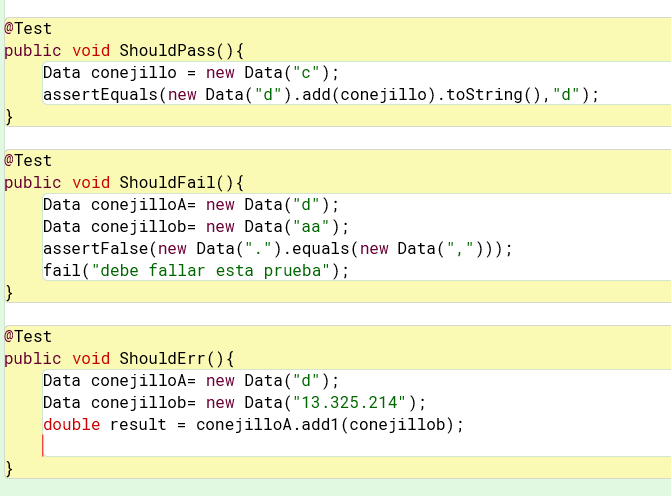
* **assertTrue(boolean condition):** Este método verifica que la condición proporcionada sea verdadera. Si la condición es verdadera, la prueba pasa satisfactoriamente; de lo contrario, la prueba falla. Es útil para verificar que ciertas condiciones o afirmaciones sean ciertas durante la ejecución de una prueba.
* **assertFalse(boolean condition):** Similar a assertTrue, pero verifica que la condición proporcionada sea falsa. Si la condición es falsa, la prueba pasa; de lo contrario, falla. Se utiliza para verificar que ciertas condiciones no sean ciertas durante la ejecución de una prueba.
* **assertEquals(expected, actual):** Este método compara dos valores: el valor esperado (expected) y el valor actual (actual). Si ambos valores son iguales, la prueba pasa; de lo contrario, falla. Es útil para verificar si un valor calculado o devuelto por un método es igual al valor que se espera en una prueba.
* **assertArrayEquals(expectedArray, actualArray):** Similar a assertEquals, pero se utiliza específicamente para comparar dos matrices. Verifica si las dos matrices tienen la misma longitud y si todos los elementos en las mismas posiciones son iguales. Si las matrices son iguales, la prueba pasa; de lo contrario, falla.
* **assertNull(object):** Este método verifica si el objeto proporcionado es nulo. Si el objeto es nulo, la prueba pasa; de lo contrario, falla. Se utiliza para verificar si un objeto devuelve un valor nulo como se espera en una prueba.
* **fail(message):** Este método fuerza que la prueba falle. Puede ser útil para indicar explícitamente que ciertas condiciones no se cumplieron o para marcar una prueba como incompleta o no implementada.



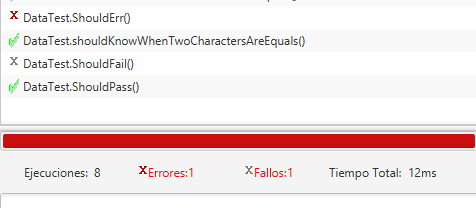
Un fallo en una prueba sucede cuando el Assert no funciona, o no hace lo que esperamos que haga, mientras que un error, sucede cuando pasa algo dentro del código, es decir un error dentro del código que estamos probando.



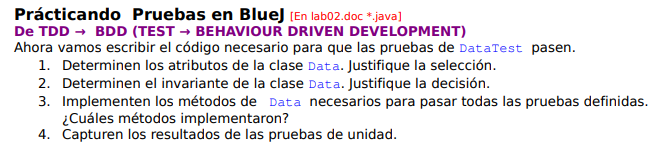
Este es un método sobre la clase Data que me suma dos Datas, pero no verifica si son de tipo n, por lo tanto, lo usamos para hacer la prueba de error



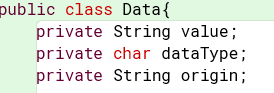
Las pruebas son hechas sobre la clase Data, el primero debe pasar, el segundo fuerza a fallo y el tercero devuelve un error



Se puede ver que el shouldErr falla, el shouldFail falla y el ShouldPass pasa



1. los atributos son:

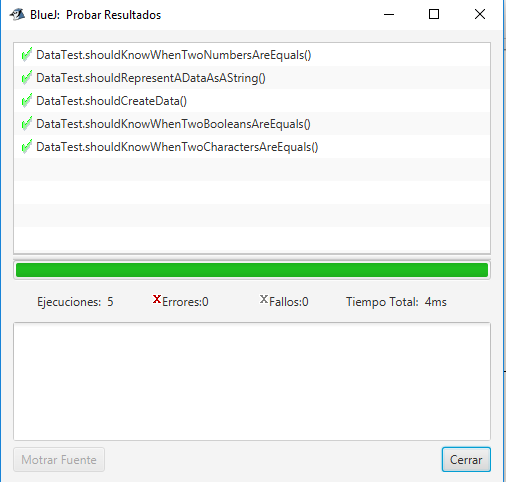


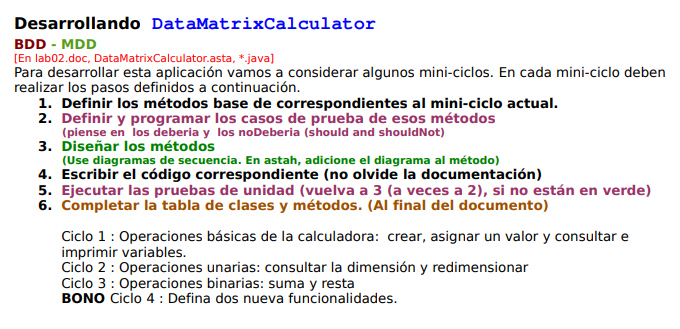
El Value guarda el valor que se va a usar, por ejemplo, si llamas una nueva instancia y pones Data(“False”), el value será “FALSE”

El dataType es el atributo que contiene el tipo de dato ‘n’,’b’,’c’

Y origin, tiene el string que le pusiste cuando creas una instancia, entonces volviendo al ejemplo inicial, el origin será “False”

1. La invariante de Data es que independientemente del valor que ingresemos al instanciar un Data, siempre tendrá asignado un value y un tipo, por ejemplo, si hago Data(“”), su value será “FALSE” y su tipo es ‘b’}
2. Implementamos los métodos del constructor, uno privado setType, addBool, addChar, addNum, subBool, subChar, subNum, add, sub, type, toString, string, equals
3. Captura:





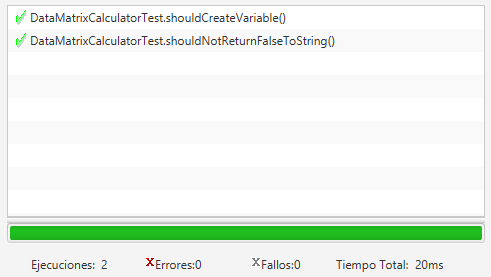
**Ciclo1:**

1. assign(String name, String values[][] ) para asignar una variable a un valor

String[] variables() para devolver todos los nombres de variables

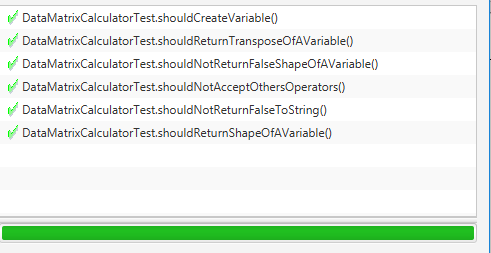
String toString(String variable) devuelve la representación de la matriz en string

1. Casos de prueba en BlueJ
2. Diagramas en astah
3. Código en BlueJ
4. Pruebas



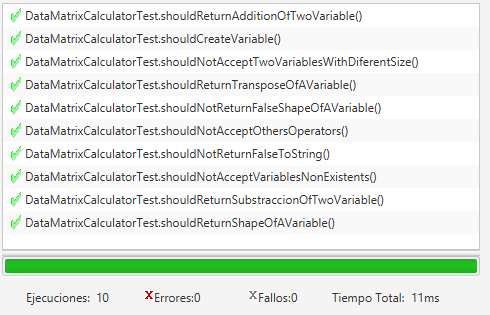
**Ciclo2:**

1. void assign(String a, char unary, String b) para asignar a “a” una operación binaria que hagamos en “b”
2. Casos de prueba en BlueJ
3. Diagramas en astah
4. Código en BlueJ
5. Pruebas



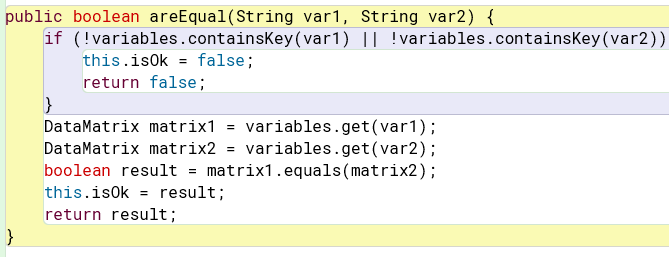
**Ciclo3:**

1. void assign(String a, String b, char sBinary, String c) donde a “a” se le asigna la operación binaria entre “b” y “c”
2. Casos de prueba en BlueJ
3. Diagramas en astah
4. Código en BlueJ
5. Pruebas



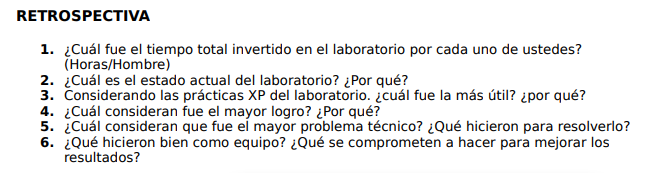
**Ciclo4:**

1. void assign(String a, String b, char sBinary, String c) donde a “a” se le asigna la operación binaria entre “b” y “c”
2. Casos de prueba en BlueJ
3. Diagramas en astah
4. Código en BlueJ
5. Pruebas

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

|  |  |  |
| --- | --- | --- |
| Ciclo | DataMatrixCalculator | DataMatrixCalculatorTest |
| 1 | assign(String name, String values[][] )  String[] variables()  String toString(String variable) | shouldCreateVariable()  shouldNotReturnFalseToString() |
| 2 | assign(String a, char unary, String b) | shouldReturnShapeOfAVariable()  shouldNotReturnFalseShapeOfAVariable()  shouldReturnTransposeOfAVariable()  shouldNotAcceptOthersOperators()  shouldNotAcceptVariablesNonExistents() |
| 3 | assign(String a, String b, char sBinary, String c) | shouldReturnAdditionOfTwoVariable()  shouldNotAcceptTwoVariablesWithDiferentSize()  shouldNotAcceptOthersOperators()  shouldNotAcceptVariablesNonExistents() |
| 4 | areEqual(String var1, String var2) | testMultiplyByScalar() |
| 5 | multiplyByScalar(String resultVar, String var, String scalar) | testAreEqual() |



1. 14 horas por cada uno
2. El laboratorio se encuentra totalmente finalizado, puesto que trabajamos desde el primer día arduamente para dejarlo en dicho estado.
3. All code must have unit test, fue el más útil de este laboratorio, en sí, el laboratorio se trababa de esa práctica xp y de enseñarnos porque las pruebas son necesarias dentro de un proyecto, pues me rectifica que nuestr código en realidad está funcionando de manera correcta
4. Nuestro mayor logro fue dejar el proyecto finalizado y haber aprendido como usar las pruebas dentro de un proyecto en java y como funciona su Framework JUnit
5. En un principio hubo varias pruebas que no querían correr de manera correcta, pero eso nos ayudó a ratificar algunos puntos erróneos dentro de la funcionabilidad del código y a solucionarlos, y haciendo precisamente eso, corrigiendo el código, solucionamos que todas las pruebas corrieran
6. Fue que nos entendimos muy bien, no hubo pleitos o desacuerdos grandes y cualquier problema que uno tuviera el otro ayudaba.