**ESCUELA COLOMBIANA DE INGENIERÍA**

**PROGRAMACIÓN ORIENTADA A OBJETOS**

**2019-02**

**Laboratorio 2/6**

**Eduard Arias – Juan Diaz**

**Conociendo el proyecto**

**1.** El proyecto BlueJ “calMat” 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 proyecto tiene una carpeta llamada “doc” con documentos, el ejecutable de BlueJ, archivos Java que representan clases de Java y un documento README.

**2.** Exploren el proyecto en BlueJ

¿Cuántas clases tiene? ¿Cuál es la relación entre ellas?

Tiene 3 clases. La relación es que una es una calculadora de matrices, otra es la clase del objeto matriz y por último tenemos la clase de pruebas de las matrices.

¿Cuál es la clase principal? ¿Cómo la reconocen?

La clase principal es Matriz ya que es la clase en donde se realizan las pruebas del proyecto.

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

La clase de pruebas, que revisa que el proyecto esté funcionando cómo debería.

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

En la clase Matriz hace falta documentación de algunos métodos ya que están sin construir.

En CalMat muestra todos los métodos, pero nada está documentado, hace falta toda la información.

**4.** Revisen el código del proyecto, ¿en qué estado está cada clase? (Detalle el estado de

codificación)

Todos los métodos están incompletos o vacíos, únicamente están enunciados.

**Ingeniería reversa**

**MDD MODEL DRIVEN DEVELOPMENT**

**1.** Genere el diagrama de clases correspondiente a CalMat con todos sus elementos. (No incluya la clase de pruebas)

En el archivo .astah

**2.** ¿Qué tipos de contenedores tienen sus colecciones? Consulte la especificación y el API Java ¿Qué diferencias hay entre ellos?

Tiene un Stack, y los arreglos int[]. Un Stack representa una colección ultimo en entrar es el primero en salir, en donde los objetos están “apilados”. Un arreglo int[] es una colección de objetos de tipo int organizados en posiciones. Se diferencian en la forma como se usan, mientras que el Stack es dinámico y el otro arreglo no.

**Conociendo Pruebas en BlueJ**

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

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

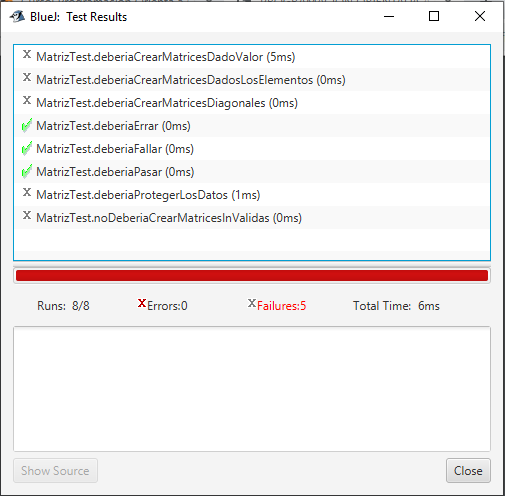
clase MatrizTest.

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

Tiene las etiquetas Before, Test y After. En total tiene 11 métodos. Tiene 9 métodos de prueba y son los que tienen la etiqueta Test.

**2.** Ejecuten los tests de la clase MatrizTest. (click derecho sobre la clase, Test All)

¿cuántos tests se ejecutan? ¿cuántos pasan las pruebas? ¿por qué?



Se ejecutan 8 test ya que a una prueba le faltaba la etiqueta Test (deberiaCrearMatricesIdentidad). Pasan 3 por que no están implementados.

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

Before: Lo que se prepara para hacer las pruebas antes de ejecutarlas.

Test: Son los métodos de pruebas

After: Termina y desarma lo que estaba preparado para terminar la ejecución.

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

assertTrue: Verifica que la condición sea verdadera

assertFalse: Verifica que la condición sea falsa.

assertEquals: Compara dos objetos o elementos y verifica que sean iguales.

assertNull: Verifica que un objeto o elemento sea nulo.

Fail: Falla la prueba.

**5.** Investiguen la diferencia que entre un fallo y un error en Junit. Escriba código usando los

métodos anteriores para lograr que los siguientes tres casos de prueba se comporten como

lo prometen deberiaPasar, deberiaFallar, deberiaErrar.

Un fallo quiere decir que todo se ejecutó bien, pero la prueba no pasó, en cambio un error quiere decir un error sintáctico en el código.



**Practicando Pruebas en BlueJ**

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

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

**1.** Determinen las estructuras de datos necesarias para almacenar los elementos de una

matriz. Justifique la selección.

Usando ArrayList para guardar y representar las matrices, porque se comportan casi igual que los arreglos, pero son mutables.

**2.** Implementen los métodos necesarios para pasar todas las pruebas definidas. ¿Cuáles

métodos implementaron?

Se implementaron los métodos:

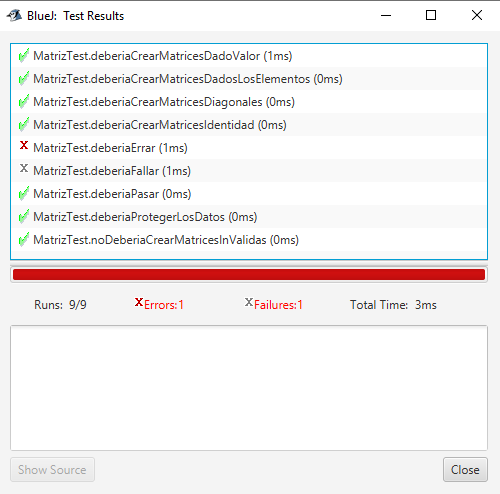
-Constructor 1

-dimension()

-get(int f, int c)

-ambos equals

-Constructor 2 y 3



**Desarrollando**

**BDD - MDD**

Para desarrollar esta aplicación vamos a considerar los siguientes ciclos de desarrollo.

* Ciclo 1: Operaciones de pila: empile, desempile y consulte
* Ciclo 2: Operaciones entre matrices: sumar, restar y multiplicar (elemento a elemento y matricial)
* Ciclo 3: Operaciones de agregación para matrices: suma, mínimo, máximo, promedio
* Ciclo 4: Operaciones de agregación por columnas o filas: suma, mínimo, máximo, promedio
* Ciclo 5: Proponga dos nuevas funcionalidades

En cada mini-ciclo deben realizar los pasos definidos a continuación.

1. Definir los métodos base de correspondientes al ciclo actual.
2. Generar y programar los casos de prueba (piense en los debería y los noDebería)
3. Diseñar los métodos (use diagramas de secuencia. En astah, adicione el diagrama al método)
4. Generar y programar los casos de prueba de los métodos de la solución (piense en todos los debería y en todos los noDebería) [OPCIONAL]
5. Escribir el código correspondiente (no olvide la documentación)
6. Ejecutar las pruebas de unidad (vuelva a 3 (a veces a 2). si no están en verde)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **CICLO** | **CalMat** | **CalMatTest** | **Matriz** | **MatrizTest** |
| **1** | Constructor  consulte()  empile(int[][])  empile(int[])  empile(int,int,int)  empile(int)  desempile() | ShouldCreate()  shouldQuery()  shouldNotQuery()  shouldStack ()  shouldNotStack()  shouldUnstack()  shouldNotUnstack() | toString() | shouldQuery() |
| **2** |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |