UNIVERSIDAD ESCUELA COLOMBIANA DE INGENIERÍA
JULIO GARAVITO

LABORATORIO 2

PROGRAMACIÓN ORIENTADA A OBJETOS

NOMBRES:

CHRISTIAN ALFONSO ROMERO MARTINEZ
ANDERSON FABIAN GARCIA NIETO

PROFESORA:
MARIA IRMA DIAZ ROZO

14/02/24

Conociendo el proyecto

1. El proyecto "graphCalculator" contiene una construcción parcial del sistema. Revisen el directorio donde se encuentra el proyecto. Describan el contenido en términos de directorios y de las extensiones de los archivos.

En el directorio GrahCalculator se cuenta con 11 archivos y un subdirectorio doc, de los archivos queremos destacar los tipo CLASS, que justamente responde a las clases con las que contamos en el proyecto, por otro lado tenemos .java, que es un archivo de origen java, también .ctxt, este tipo de archivo se utiliza para almacenar datos en texto plano, por último tenemos un .txt, en el readme que es donde se da una identificación del proyecto. Al ingresar al subdirectorio se encuentran 5 archivos html que hacen referencia a la documentación de cada clase/metodo, tambien hay archivos .txt que que hacen referencia a un archivo de registro (log) que suele generarse automáticamente por programas, aplicaciones o sistemas para guardar información sobre eventos, errores, advertencias o actividades que ocurren durante la ejecución de un proyecto o software, por último se observan archivos .list siendo el lugar donde se guardan datos de manera estructurada.

2. Exploren el proyecto en BlueJ

¿Cuántas clases tiene?

3 clases, siendo el grafo, la calculadora del grafo y la clase de prueba unitaria

¿Cuál es la relación entre ellas?

Cómo GraphCalculator depende de la clase Graph, esta es una relación de tipo composición debido a que hay una clase que usa otras clases para compartir la información, lo mismo sucede con la clase de pruebas GraphTest.

¿Cuál es la clase principal de la aplicación?

La clase principal de la aplicación es la clase Graph debido a que las otras clases hacen uso de ella

¿Cómo la reconocen?

La relación entre GraphTest y Graph es una relación de uso, puesto que los métodos de GraphTest llaman a los de Graph para poder llevar a cabo su tarea, por otro lado GrahCalculator usa a Graph de forma estructural.

¿Cuáles son las clases "diferentes"?

La clase diferente hace referencia a la clase que aparece con color verde en BlueJ, siendo la clase con nombre GraphTest, esta clase se representa de dicha forma, debido a que va a contener las pruebas para las otras clases, para confirmar el

funcionamiento del código en las otras clases, debido a que pueden existir cambios en este.

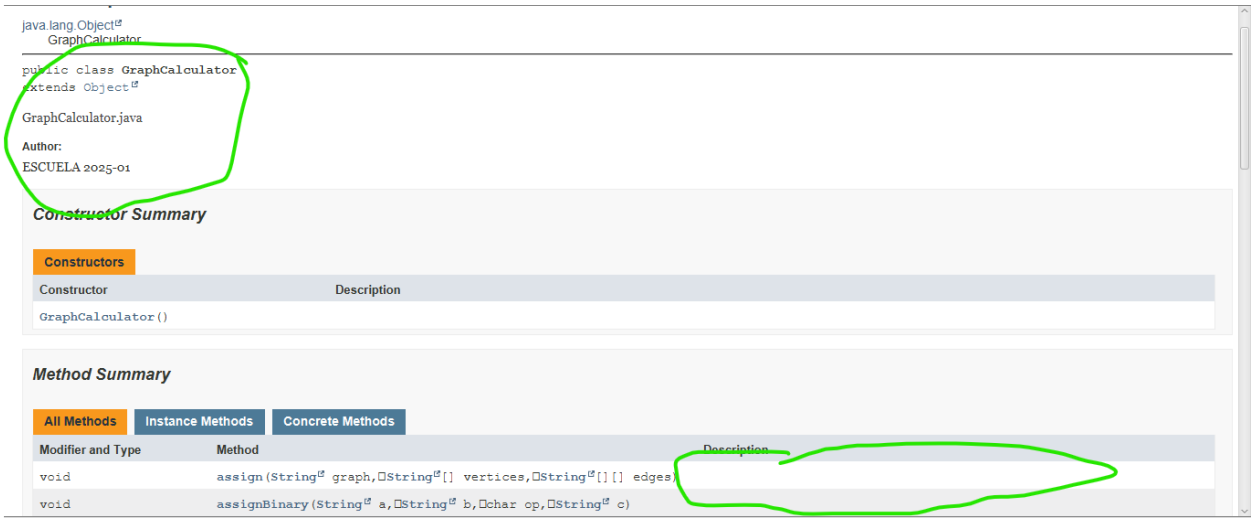
¿Cuál es su propósito?

Retomando lo anterior, el propósito de la clase de pruebas es comprobar que el código hace lo que debe hacer, todo esto debido a que existe la posibilidad de que se modifique el código de algún método en las clases por algún error. Básicamente confirma que la clase se compila de una forma correcta.

Teniendo en cuenta las clases sin incluir a la de pruebas se responden las siguientes preguntas:

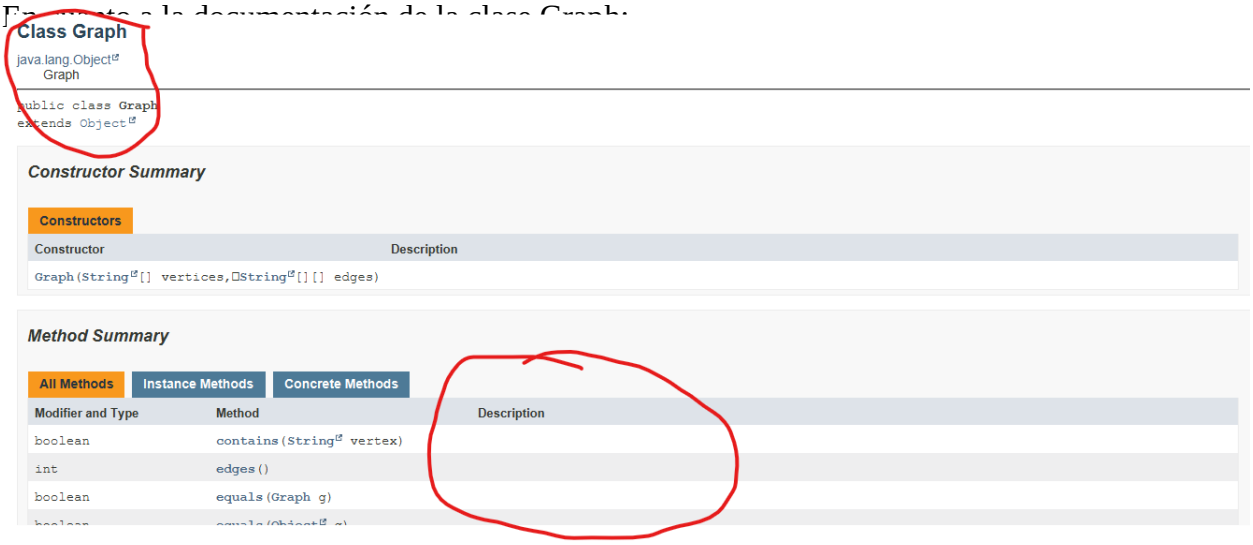
3. Generen y revisen la documentación del proyecto: ¿Está completa la documentación de cada clase? (Detallen el estado de documentación: encabezado y métodos)

En la clase GraphCalculator, se observa lo siguiente en la documentación.



De esta imagen se puede observar que no existe un encabezado claro, debido a que no se especifica que es lo que hace la clase, en cuanto a los métodos no existe una descripción de los métodos que se crean, analizando lo ocurrido esto sucedió debido a que se en vez de escribirse la descripción como documentación, se escribió como comentario

En cuanto a la documentación de la clase Graph:



Class Graph

java.lang.Object
Graph

public class Graph
extends Object

Constructor Summary

Constructors

Constructor	Description
Graph(String[] vertices,String[][] edges)	

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method	Description
boolean	contains(String vertex)	
int	edges()	
boolean	equals(Graph g)	

Se observa un caso parecido que con la clase anterior, debido a que no existe una debida descripción de la clase, ni de la descripción de los métodos, o las entradas, esto porque no se escribió correctamente la documentación de la clase y la descripción de los métodos no está escrita de ninguna manera, a diferencia de la clase anterior que se había escrito la documentación a través de un comentario.

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 diferencia hay entre el código, la documentación y los comentarios?

La clase Graph tiene varios métodos que devuelven diferentes valores, pero la mayoría de ellos están implementados de manera muy básica, solo con un return sin mucha lógica adicional. Solo el último método tiene algún tipo de comentario o explicación. En cuanto a la documentación, todos los métodos están correctamente descritos, diferenciando claramente entre los constructores y los demás métodos.

Por otro lado, la clase GraphCalculator tiene todos sus métodos comentados. En el constructor, hay una estructura definida, lo que contrasta con los otros métodos, que están vacíos o solo tienen un return sin mayor funcionalidad. La documentación en esta clase también está presente, pero parece que falta implementar la lógica en la mayoría de los métodos.

INGENIERÍA EN REVERSA

1. Realicen el diagrama de clases correspondiente al proyecto. (No incluyan la clase de pruebas)

Nota: Está en el Astah

2. ¿Cuáles contenedores están definidos? ¿Qué diferencias hay entre el nuevo contenedor, el ArrayList y el vector [ ] que conocemos? Consulte el API de java.

Los contenedores definidos son:

- vector []

-vectorDentroVector[][]

-TreeMap<>

-El ArrayList es importado en el código de la clase Graph, aunque no es usado

Diferencias entre el Nuevo Contenedor (Tree Map), ArrayList y el vector[]

Característica	TreeMap	ArrayList	Vector
Propósito	Implementa una estructura de datos de mapa ordenado	Implementa una lista dinámica	Implementa una lista dinámica, es similar a ArrayList
Ordenación	Mantiene las entradas ordenadas según las claves	No garantiza un orden específico	No garantiza un orden específico
Sincronización	No está sincronizado	No está sincronizado	Está sincronizado

Rendimiento	Más lento para operaciones de inserción y eliminación	Rápido para operaciones de acceso aleatorio	Más lento que ArrayList para operaciones de acceso
Implementación	Basado en un árbol rojo-negro	Basado en una matriz redimensionable	Basado en una matriz redimensionable
Uso común	Cuando se necesita un mapa ordenado	Cuando se necesita una lista dinámica no sincronizada	Cuando se necesita una lista dinámica sincronizada

3. En el nuevo contenedor

¿Cómo adicionamos un elemento?

treeMap.put(“elemento”,#);

es el valor de nuestro elemento, por ejemplo podemos guardar como elemento libros y en el valor la cantidad de ellos.

¿Cómo lo consultamos?

treeMap.get(elemento)

Integer variableGuardar = treeMap.get("elemento");

variableGuardar es donde vamos a almacenar el valor de elementos que tenemos, para luego imprimirlo de forma tradicional.

¿Cómo lo eliminamos?

treeMap.remove(elemento)

Integer valorEliminar = treeMap.remove("elemento");

valorEliminar nos ayuda a almacenar el valor asociado a la clave eliminada

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 GraphTest

1. **Revisen el código de la clase GraphTest.**

   ¿Cuáles etiquetas tiene (componentes con símbolo @)?

   @Before

   @Test

   @After

   ¿Cuántos métodos tiene?

   La clase GraphTest cuenta con 8 métodos.

   ¿Cuántos métodos son de prueba?

   6 métodos son de prueba.

   ¿Cómo los reconocen?

   Gracias a las etiquetas, se permite identificar de forma netamente descriptiva cada método.

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

   ¿Cuántas pruebas se ejecutan?

   Las 6 pruebas se ejecutaron.

   ¿Cuántas pasan?

Solo paso 1 prueba.

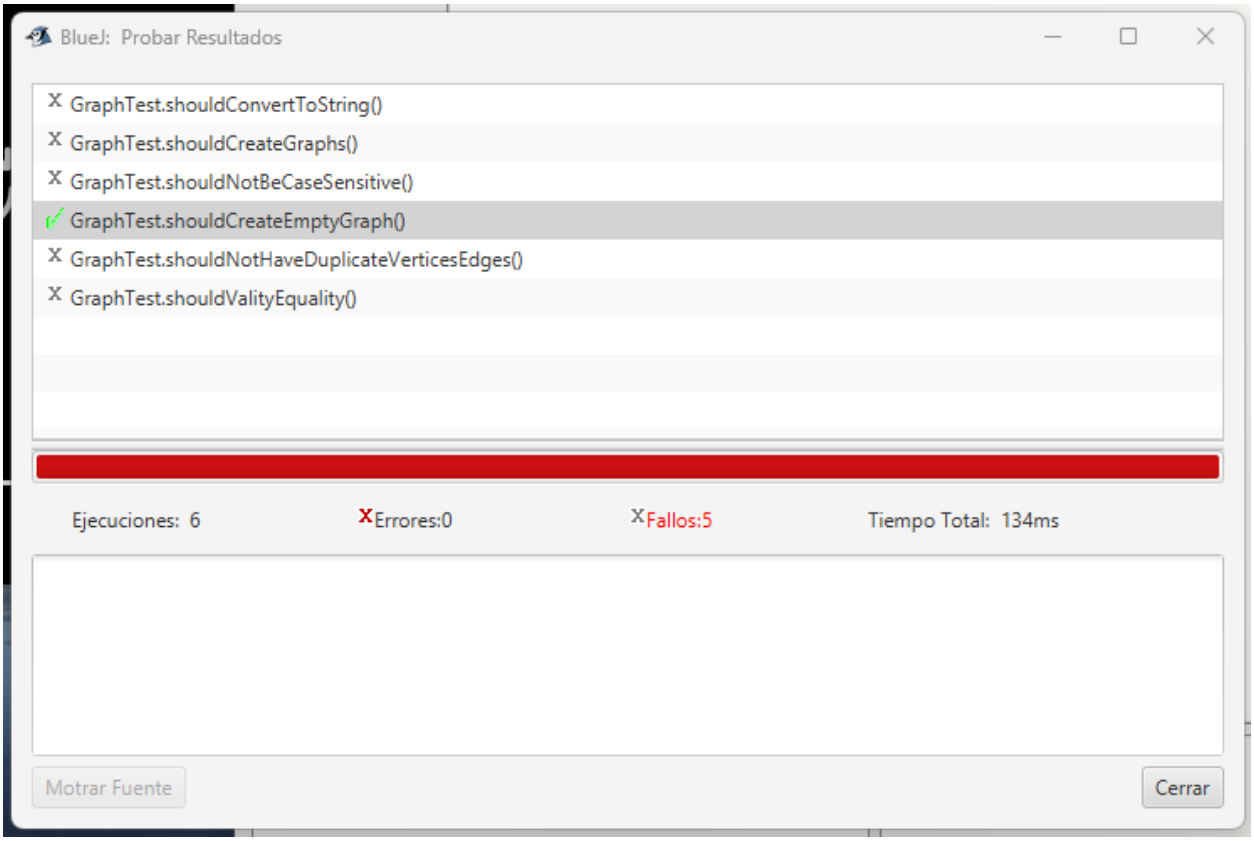
ShouldCreateEmptyGraph

¿Por qué?

Esta prueba pasa porque su lógica no está bien definida

Debido a que en Graph retorna para edge y vértices 0 hace que siempre se cumplan los deberian de la prueba.

Capturen la pantalla.



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

@Before

Esta etiqueta quiere decir antes, antes de que cada prueba se ejecute, su responsabilidad es denotar el fragmento del código que inicializa para

asegurar que los recursos estén disponibles para el correcto funcionamiento de las pruebas.

@Test

Este nos menciona que el código a continuación es una prueba unitaria, esto es muy importante en JUnit puesto que él lo diferencia para ejecutar estos métodos.

@After

Este nos nombra la parte del código que va a liberar los recursos inicializados en before y usados en test.

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

AssertTrue

Este se encarga de verificar que sea TRUE una condición para que la prueba pase, de lo contrario no pasa.

Ejemplo de lo anterior es

assertTrue(número > 0) para asegurarnos que “número” sea positivo

AssertFalse

Al igual que el método anterior se encarga de validar el valor de verdad de una verificación, en este caso si es FALSE, la prueba pasa, de lo contrario no.

Ejemplo de lo anterior es

assertFalse(número < 0) Para validar que el número no sea menor que 0

AssertEquals

Este válida el valor entre 2 valores para definir si pasa o no la prueba

Ejemplo de lo anterior es

assertEquals(return, numero) Este nos permite validar si lo que nos retorno algún método con el número son iguales.

AssertNull

Este válida que un objeto tenga como valor nulo, si llega a tener un valor nulo la prueba pasa, de lo contrario no

Ejemplo de lo anterior es

assertNull(return) Se valida si lo retornado de algun metodo es nulo

Fail

A diferencia de los métodos anteriores, esta fuerza un fallo si o si, se suele usar para cuando la prueba fue muy lejos en el codigo y debio haber terminado antes

Ejemplo de lo anterior

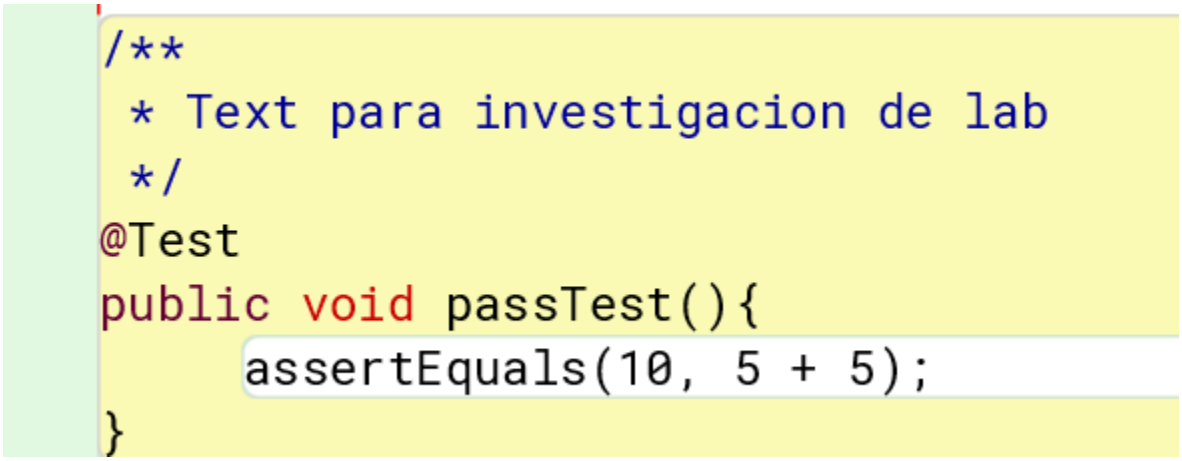
fail() y se coloca en un punto del código donde la prueba no debería ejecutarse

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

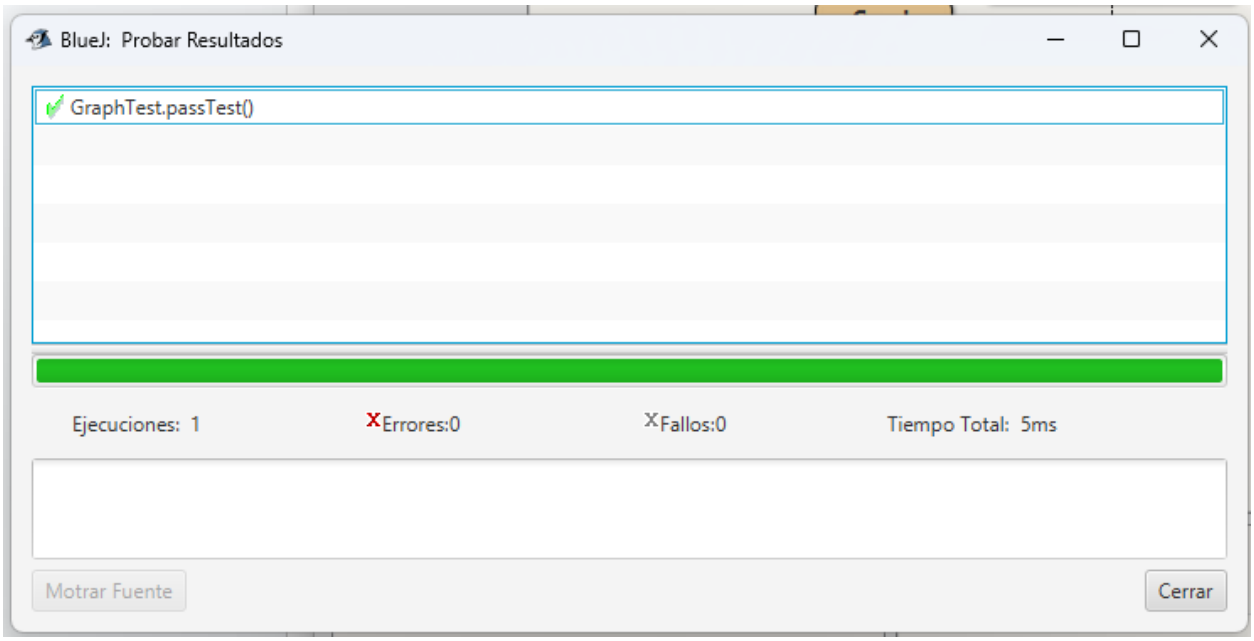
Aspecto	Fallo	Error
Ocurre cuando	Una definición assert no se cumple.	Una excepción inesperada se lanza durante la ejecución de la prueba.
Causa	Hay una correcta ejecución pero el resultado no coincide con el debería.	El código presenta problemas como ejecuciones inesperadas

Ejemplos	assertEquals(10, 5 + 4)	Acceder a array[5] en un array de longitud 3
Explicacion	El resultado del código no funciona como lo esperado	Hay un error en la ejecucion del codigo que debe presentar correcciones
Metodos usados	assertTrue asserFalse asserEquals assertNull	NullPointerException División por cero Y en general excepciones no controladas
JUnit	□ Fallo	✕Error

shouldPass

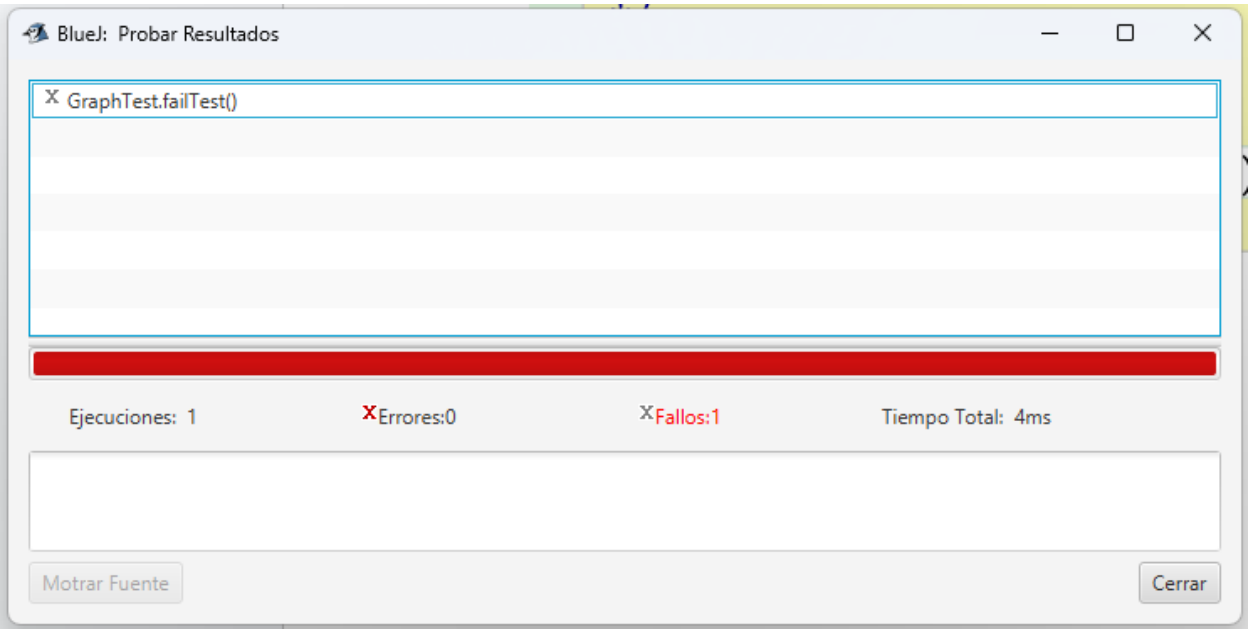


```
/**
 * Text para investigacion de lab
 */
@Test
public void passTest(){
      assertEquals(10, 5 + 5);
}
```



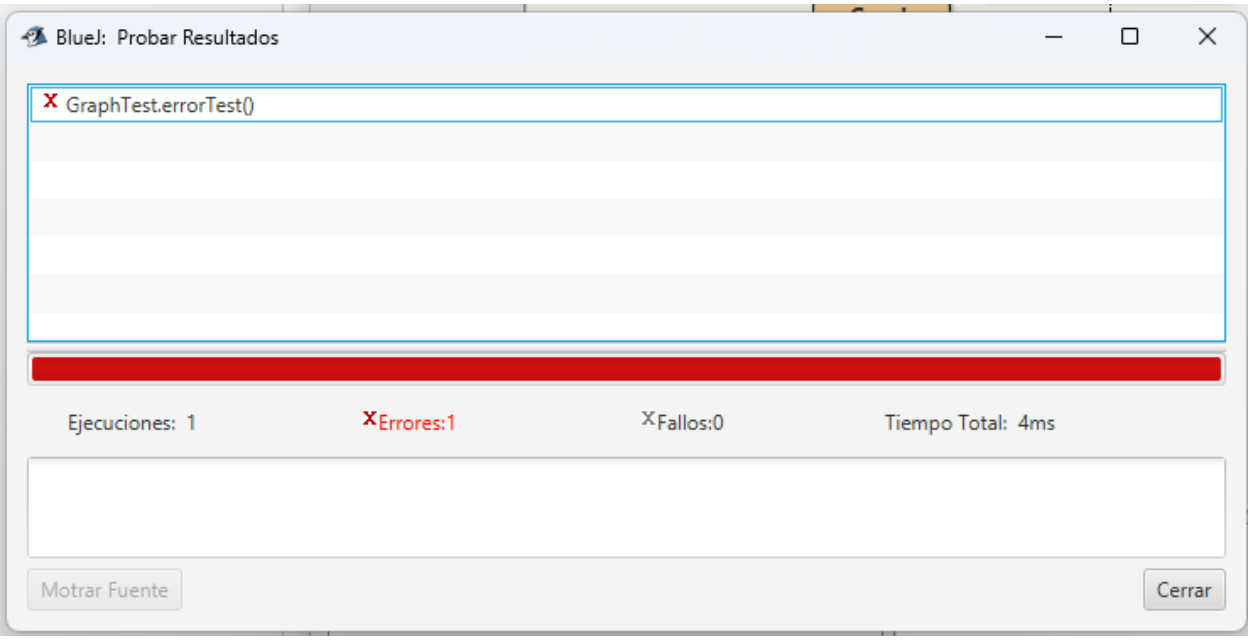
shouldFail

```
/**
 * Text para investigacion de lab
 */
@Test
public void failTest(){
     assertEquals(10, 5 + 4);
}
```



shouldError

```java
/**
 * Text para investigacion de lab
 */
@Test
public void errorTest() {
int[] array = {1, 2, 3};
int valor = array[5];
}
```



Prácticando Pruebas en BlueJ

De TDD → BDD (TEST → BEHAVIOUR DRIVEN DEVELOPMENT)

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

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

Se determinó solamente agregar dos atributos, el atributo vértices y edges siendo de tipo Set.

Se escogió que el tipo de los vértices y edges fuera Set por las propiedades que incorpora la librería Set (Conjuntos en español), debido a que varias de las características que tienen los conjuntos, siendo un ejemplo que no existen elementos repetidos, o no existe el orden. Por otra parte también se incorporó debido a la existencia de la función .size() que devuelve la cantidad de elementos en un conjunto.

2. Determinen el invariante de la clase Graph. Justifique la decisión.

En este proyecto existen varios invariantes notables, algunos de ellos no venían incluidos en la especificación de lo que se quería que hiciera el método, estos invariantes son:

- La creación de las aristas siempre deben conectarse con vértices existentes.
- Las aristas se almacenan en un formato definido, siendo este por orden alfabético.
- Los vértices se almacenan con letras mayúsculas.

- Un arco por definición está conectado con su inverso.

Todos estos son invariantes para el proyecto, el que más concuerda con la definición es el último, debido a que es algo que se sabe antes, que funciona durante y también al final, debido a que si se asigna un nuevo grafo con mínimo un arco está conectado con su inverso.

Ejemplo:

(A,B) -≡ (B,A)

3. Implementen los métodos de Graph necesarios para pasar todas las pruebas definidas.

¿Cuáles métodos implementaron?

Se implementaron los métodos:

public Graph(String[] vertices, String[][] edges)

private void addEdge(String verticeUno, String verticeDos)

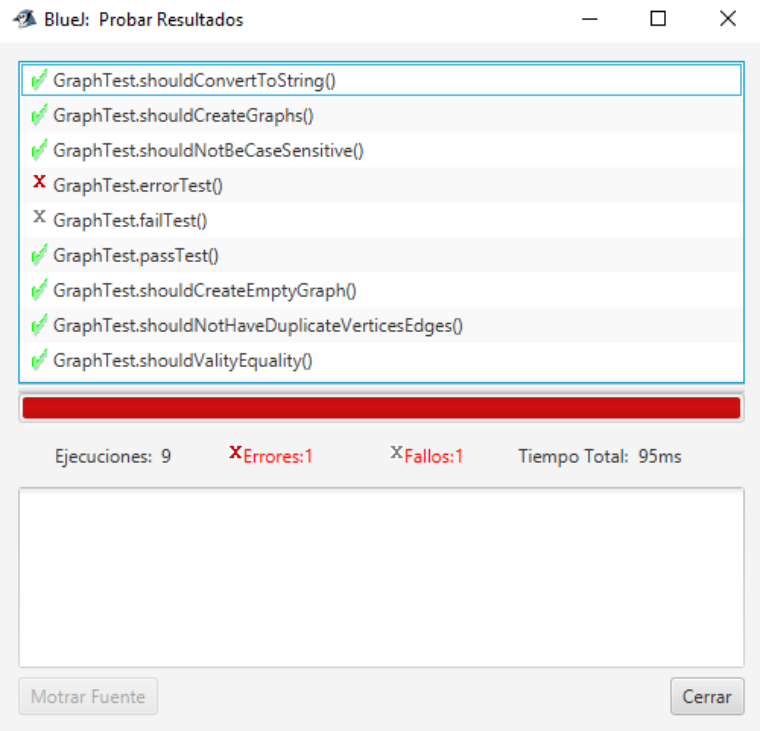
public boolean contains(String vertex)

public boolean equals(Object obj)

public String toString()

4. Capturen los resultados de las pruebas de unidad.

Nota: De acuerdo al código proporcionado, son 6 pruebas de unidad, en la imagen salen 9, ya que anteriormente se pide que se creen pruebas unitarias, por eso salen 3 más, de las cuales 2 si o si deben fallar o devolver un error.



DESARROLLANDO

Ciclo 1 : Operaciones básicas de la calculadora: crear una calculadora y asignar y consultar un grafo

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

1. **Constructor:** Para crear una instancia de la calculadora.
2. **Asignar un grafo:** Asignar un grafo a una variable con un nombre específico.
3. **Consultar un grafo:** Obtener la cantidad de vértices, la cantidad de aristas y la representación en cadena del grafo.

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

1. **shouldAssignGraphToVariable():** Prueba que se pueda asignar un grafo a una variable en la calculadora.
2. **shouldNotAssignGraphWithNullName():** Prueba que no se pueda asignar un grafo a una variable con nombre nulo.
3. **shouldReturnCorrectGraphInfo():** Prueba que se devuelva la información correcta del grafo al consultar una variable.
4. **shouldNotReturnInfoForNonExistentVariable():** Prueba que no se devuelva información para una variable que no existe.

3. Diseñar los métodos (Astah)

4. Escribir el código correspondiente (BlueJ)

5. Ejecutar las pruebas de unidad

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

Ciclo 2: Operaciones unarias: insertar y eliminar arcos; consultar si un conjunto de vértices pertenece al graph y retornar el camino que pasa por un conjunto de vértices

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

1. assignUnary: Realiza operaciones unarias sobre un grafo almacenado, utilizando dos vértices y un operador.
Las operaciones incluyen insertar una arista, eliminar una arista, verificar la pertenencia de vértices y encontrar un camino entre dos vértices.

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

1. shouldDeleteEdge(): Prueba que se pueda eliminar una arista existente en el grafo.
2. shouldNotDeleteNonExistentEdge(): Prueba que no se pueda eliminar una arista que no existe en el grafo.
3. shouldDeleteEdgeRegardlessOfOrder(): Prueba que se pueda eliminar una arista independientemente del orden de los vértices.

3. Diseñar los métodos (Astah)

4. Escribir el código correspondiente (BlueJ)

5. Ejecutar las pruebas de unidad

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

Ciclo 3: Operaciones binarias: union, intersección, diferencia y junta

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

1. assignBinary(): Realiza una operación binaria sobre dos grafos y asigna el resultado a una variable.

Las operaciones incluyen la unión, intersección, diferencia y junta.

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

1. shouldPerformUnion(): Prueba que se pueda realizar la unión de dos grafos.
2. shouldPerformIntersection(): Prueba que se pueda realizar la intersección de dos grafos.
3. shouldPerformDifference(): Prueba que se pueda realizar la diferencia de dos grafos.
4. shouldPerformJoin(): Prueba que se pueda realizar la junta de dos grafos.

3. Diseñar los métodos (Astah)

4. Escribir el código correspondiente (BlueJ)

5. Ejecutar las pruebas de unidad

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

Ciclo 4: Operaciones extras: Comparar dos grafos y retorna el grafo más largo, revisar si un grafo es convexo.

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

1.compareGraphs(): Compara dos grafos por la cantidad de aristas que poseen.
2.isConnected(): Verifica si un grafo es conexo, esto quiere decir que todos los vertices tienen un arco entre todos.

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

1.shouldReturnLargerGraph(): Prueba que devuelve el grafo más grande al comparar dos grafos.
2.shouldReturnEqualGraphs(): Prueba que se devuelva "Iguales" al comparar dos grafos con el mismo número de aristas.
3.shouldBeConnected(): Prueba que un grafo sea conexo.
4.shouldNotBeConnected(): Prueba que un grafo no sea conexo.

3. Diseñar los métodos (Astah)

4. Escribir el código correspondiente (BlueJ)

5. Ejecutar las pruebas de unidad

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

Clases y Métodos

Ciclo	GraphCalculator	GraphCalculatorTest
1 Operaciones básicas	String consult(String nombre)	shouldAssignGraphToVariable()
	void assign(String nombre, String[] vertices, String[][] edges)	shouldNotAssignGraphWithNullName()
		shouldReturnCorrectGraphInfo()
		shouldNotReturnInfoForNonExistentVariable()
2 Operaciones unarias	void assignUnary(String graphName, String vertexA, String vertexB, char op)	shouldDeleteEdge()
		shouldNotDeleteNonExistentEdge()
		shouldDeleteEdgeRegardlessOfOrder()
3 Operaciones binarias	void assignBinary(String a, String b, char op, String c)	shouldPerformUnion()
		shouldPerformIntersection()
		shouldPerformDifference()
		shouldPerformJoin()
4 Métodos adicionales	boolean isConnected(String graphName)	shouldBeConnected()
	String compareGraphs(String graphName1, String graphName2)	shouldNotBeConnected()
		shouldReturnLargerGraph()
		shouldReturnEqualGraphs()

RETROSPECTIVA

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

(Horas/Nombre)

Anderson Fabian Garcia Nieto 15 horas de trabajo

Christian Alfonso Romero Martinez 12 horas de trabajo

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

El estado actual del laboratorio está terminado satisfactoriamente. Lo terminamos y opinamos que es un muy buen trabajo, debido a que le dedicamos el tiempo necesario para que así lo fuera

3. Considerando las prácticas XP del laboratorio. ¿Cuál fue la más útil? ¿Por qué?

Consideramos que la más útil para este laboratorio fue la práctica XP "All code must have unit tests. " debido a que a medida que se codifican y se necesitaba probar el código para verificar su correcto funcionamiento, la creación de los grafos uno por uno era muy tedioso, en cambio al crear la prueba relacionada al método facilitaba en gran medida el tiempo necesitado para verificar su funcionamiento

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

El mayor logro consideramos que es de nuevo la comunicación y el trabajo en equipo, puesto que hubo colaboración mutua y fortalecimiento de los en conceptos, que en una primera instancia no quedaron claros.

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

Uno de los mayores problemas fue la creación del método difference, debido a que se había programado con un if siendo que en su condición estaba con un o, debido a esto pasaban varios casos y al final no pasaba su prueba correspondiente, debido a este problema se nos fue mucho tiempo.

Lo que se hizo para resolverlo fue analizar el código línea por línea, dando con el error.

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

Como equipo buscamos varias maneras para despejar nuestras dudas, sabiendo que no conocíamos la manera exacta de realizar un diagrama de secuencias.

Nos comprometemos a llevar más preguntas a la profesora de la teoría, para resolver las dudas que surjan y así poder tener un mayor rendimiento en la presentación del laboratorio.

7. ¿Qué referencias usaron? ¿Cuál fue la más útil? Incluyan citas con estándares adecuados.

GeeksforGeeks. (2021, 22 junio). *What is class Invariant*. GeeksforGeeks. https://www-geeksforgeeks-org.translate.goog/what-is-class-invariant/?_x_tr_sl=en&_x_tr_tl=es&_x_tr_hl=es&_x_tr_pto=sge#:~:text=is%20class%20Invariant-,Overview%20:An%20invariant%20in%20Object%2DOriented%20programming%20refers%20to%20some,member%20(mutator)%20method%20call%20to

IBM Developer. (s. f.). https://developer.ibm.com/articles/the-sequence-diagram/

Overview (JUniT 5.12.0 API). (s. f.). https://junit.org/junit5/docs/current/api/