

LABORATORIO # 2 POOB
DISEÑO Y PRUEBAS, INTERACCION ENTRE OBJETOS

PRESENTADO A:

MARIA IRMA DIAZ ROSO

PRESENTADO POR:

CAROLINA CEPEDA

JUANITA RUBIANO

UNIVERSIDAD ESCUELA COLOMBIANA DE INGENIERIA JULIO GARAVITO

BOGOTA D.C

2025-1



OBJETIVOS

Desarrollar competencias básicas para:

1. Desarrollar una aplicación aplicando BDD y MDD.
2. Realizar diseños (directa e inversa) utilizando una herramienta de modelado ([astah](#))
3. Manejar pruebas de unidad usando un framework ([junit](#))
4. Apropiar nuevas clases consultando sus especificaciones ([API java](#))
5. Experimentar las prácticas XP : **Designing** Use [CRC cards](#) for design sessions. **Testing** All code must have [unit tests](#).

ENTREGA

- ✓ Incluyan en un archivo **.zip** los archivos correspondientes al laboratorio. El nombre debe ser los dos apellidos de los miembros del equipo ordenados alfabéticamente.
- ✓ Deben publicar el avance (al final de la sesión) y la versión definitiva (en la fecha indicada) en los espacios preparados para tal fin

CONTEXTO

Objetivo

En matemáticas discretas, en particular en teoría de grafos, un **grafo** es una estructura que consiste en un conjunto de objetos donde algunos pares de objetos están relacionados. Los objetos son llamados vértices y cada uno de los pares de vértices relacionados se llama arco.

DESARROLLO DEL LABORATORIO.

Conociendo el proyecto [En lab02.doc]

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.
2. Exploren el proyecto en BlueJ
 - ¿Cuántas clases tiene? ¿Cuál es la relación entre ellas?
 - ¿Cuál es la clase principal de la aplicación? ¿Cómo la reconocen?
 - ¿Cuáles son las clases "diferentes"? ¿Cuál es su propósito?

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: encabezado y métodos)
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?

1. En la carpeta del proyecto encontramos las siguientes extensiones:

- .java que son los archivos fuente de origen java de las clases.
- .class que contiene las clases del código compilado en java.
- CTXT que son los archivos que BlueJ crea de forma automática durante la compilación de un proyecto y contiene, a diferencia de los archivos .class , comentarios y la documentación de los métodos en el código.

- BlueJ que es el paquete del proyecto en BlueJ y al seleccionarlo abre automáticamente el proyecto en la aplicación.

A su vez. Encontramos otra carpeta llamada “doc” en la que se encuentran documentos html que contienen la documentación del proyecto. También hay un archivo tipo gif que contiene la imagen de una línea usada en diagramas de clase para simbolizar una relación.

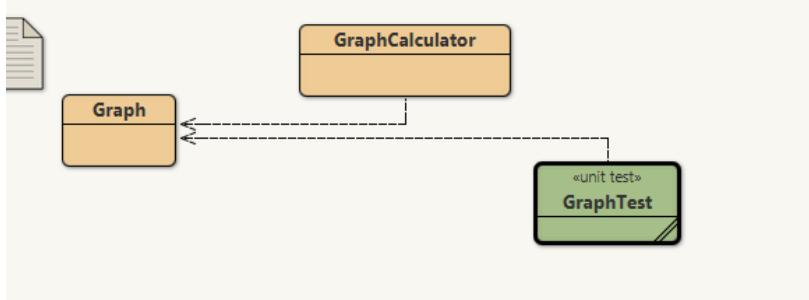
2.

¿Cuántas clases tiene?

Cuenta con tres clases: Graph, GraphCalculator y GraphTest.

¿cuál es la relación entre ellas?

Son relaciones unidireccionales, entre Graph con GraphCalculator y, Graph con GraphTest.



¿Cuál es la Clase principal de la aplicación? ¿Comó la reconocen?

La clase principal es GraphCalculator y la reconocemos porque es la que ofrece los servicios de calculadora de grafos.

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

La clase diferente es GraphTest, ya que es una clase de pruebas, cuyo propósito es probar la funcionalidad del código.

3. Generen y revisen la documentación del proyecto: ¿está completa la documentación de cada clase?

En el caso de la clase Graph, no hay ninguna clase de documentación en el código. Por otro lado, la clase GraphCalculator tiene la documentación en forma de comentarios. En el caso de GraphTest encontramos una documentación completa.

4. ¿en qué estado está cada clase?

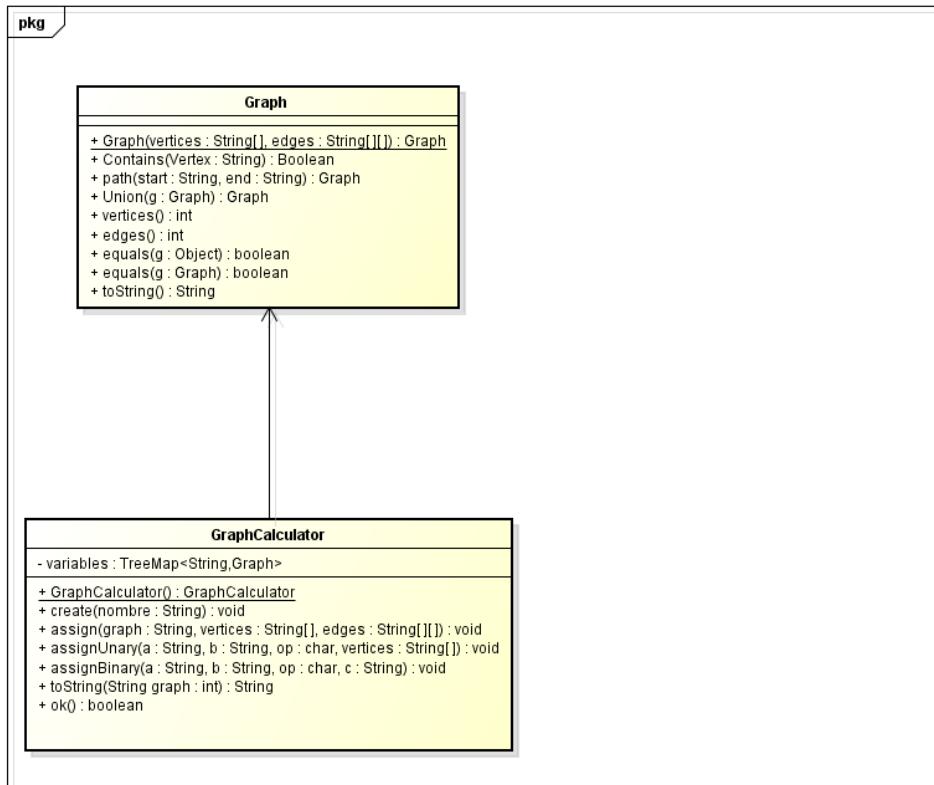
- Clase Graph: la clase no tiene atributos y tiene 8 métodos públicos.
- Clase GraphCalculator: La clase contiene un atributo privado y tiene 6 métodos públicos.
- Clase GraphTest: La clase no contiene atributos y contiene 8 métodos públicos.

Ingeniería reversa [En lab02.doc GraphCalculator.asta]

MDD MODEL DRIVEN DEVELOPMENT

1. Realicen el diagrama de clases correspondiente al proyecto. (No incluyan la clase de pruebas)
2. ¿Cuáles contenedores están definidos? ¿Qué diferencias hay entre el nuevo contenedor, el `ArrayList` y el vector `[]` que conocemos? Consulte el API de java.
3. En el nuevo contenedor, ¿Cómo adicionamos un elemento? ¿Cómo lo consultamos? ¿Cómo lo eliminamos?

1.



2.

¿Cuántos contenedores están definidos?

Son tres contenedores; vectores, ArrayList<> y el TreeMap.

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

	ArrayList	Vector
Crecimiento dinámico	Crece un 50%	Crece en un 100%
Rendimiento	Mayor rendimiento, debido a la falta de sincronización	Menor sincronización
Sincronización	NO	SI

3. En el nuevo contenedor, ¿cómo adicionamos un elemento?,¿Como lo consultamos?,¿Como lo eliminamos?

-TreeMap:

- Adicionar un elemento, usamos el metodo .put(clave, valor)
- Consultar, usamos el metodo .get(clave), si la clave no existe retorna Null.
- Eliminar, usamos el metodo .remove(clave) este solo elimina el valor, si queremos conocer el valor eliminado, usamos .remove(clave,valor)

- ArrayList< >:

- Para adicionar un elemento, se puede hacer al final del ArrayList con el metodo .add(elemento) pero si queremos aclararlo en una posicion especifica, se usa .add(indice, elemento)
- Consultar, Usamos el metodo .get(indice)
- Eliminar, usamos el metodo .remove(indice)

Para adicionar, consultar y eliminar en un vector se usan los mismo metodos de un ArrayList.

Conociendo Pruebas en BlueJ [En lab02.doc *.java]

De TDD → BDD (TEST → BEHAVIOUR DRIVEN DEVELOPMENT)

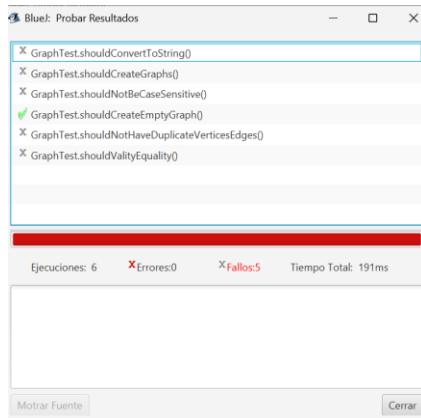
Para poder cumplir con la prácticas XP vamos a aprender a realizar las pruebas de unidad usando las herramientas apropiadas. Para eso implementaremos algunos métodos en la clase `GraphTest`

1. Revisen el código de la clase `GraphTest`. ¿cuáles etiquetas tiene (componentes con símbolo @)? ¿cuántos métodos tiene? ¿cuantos métodos son de prueba? ¿cómo los reconocen?
2. Ejecuten los tests de la clase `GraphTest`. (click derecho sobre la clase, Test All) ¿cuántas pruebas se ejecutan? ¿cuántas pasan? ¿por qué? Capturen la pantalla.
3. Estudie las etiquetas encontradas en 1 (marcadas con @). Expliquen en sus palabras su significado.
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.
5. Investiguen y expliquen la diferencia que 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`, `shouldErr`.

1.

- ¿cuáles etiquetas tiene? Contiene tres etiquetas; “Before”, “Test” y “After”.

- ¿cuántos métodos tiene y cuáles son de prueba? Tiene 8 métodos y 6 de estos son de prueba.
 - ¿Cómo los reconocen? Se reconocen debido a su etiqueta @Test y a que el comienzo de estos métodos siempre comienza por “Should” .
2. Se ejecutan 6 pruebas y solo pasa una, ya que los retornos de los métodos de la clase Graph no están definidos correctamente, en el caso del método toString encontramos que retorna un String vacío y, en otros métodos, se retorna cero, nulo, etc. La prueba de creación de un grafo vacío observamos que pasa debido a que el constructor de la clase Graph crea, por defecto, un grafo vacío.



3. @Before es utilizado antes de cada prueba para acomodar todo lo necesario para cada prueba, @Test para mostrar cuales son los métodos de prueba y @after es utilizado después de cada test para desmontar, por así decirlo, todo lo hecho y que puedan hacerse la siguiente prueba sin ninguna clase de cambio en el contexto.
4. métodos:
- assertTrue: Afirma que una condición es verdadera.
 - assertFalse: Afirma que una condición es falsa.
 - assertEquals : Afirma que dos objetos son iguales
 - assertNull: Afirma que un objeto es nulo.
 - Fail: Muestra la falla de una prueba, con o sin mensaje.

5. La diferencia entre un fallo y un error en Junit es que el fallo se da debido a que el resultado fue diferente al esperado, mientras que un error representa uno o más problemas que se encontraron al llevar a cabo la prueba y que no permitieron llegar a un resultado dado.

```
@Test  
public void shouldPass(){  
    assertTrue(true);  
}  
  
@Test  
public void shouldFail(){  
    fail();  
}  
  
@Test  
public void shouldErr(){  
    String hola="";  
    hola.substring(1,1);  
}
```

GraphTest.shouldErr()
GraphTest.shouldConvertToString()
GraphTest.shouldCreateGraphs()
GraphTest.shouldFail()
GraphTest.shouldPass()

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

De TDD → BDD (TEST → BEHAVIOUR DRIVEN DEVELOPMENT)

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

1. Determinen los atributos de la clase `Graph`. Justifique la selección.
2. Determinen el invariante de la clase `Graph`. Justifique la decisión.
3. Implementen los métodos de `Graph` necesarios para pasar todas las pruebas definidas.
¿Cuáles métodos implementaron?
4. Capturen los resultados de las pruebas de unidad.

1. Los atributos de la Clase Graph son: justifique.

Al constructor le llega como parámetro Vértices: String [] y edges: String[][][], pero para el desarrollo de código se declaran los atributos de la clase como:

- Vértices, en una lista (ArrayList) que almacena el nombre de los vértices.
- Edges, Como una lista de listas (ArrayList< ArrayList<String>>) que almacena las aristas. Cada arista es una lista con dos elementos, los vértices que conecta.

2. Determine el invariante de la clase Graph, justifique.

El número de vértices y aristas no cambia sin una operación válida.

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

- Se instanciaron los atributos, se completó el constructor con:
 - El cambio de tipo, pasamos de tener un vector de vértices a tener un ArrayList<String> de vértices y de tener un vector de dos dimensiones como edges, a tener un ArrayList<ArrayList<String>>, esto para trabajar con más facilidad.
 - Al cambio de tipo va ligado un condicional que se asegura de que no se tengan elementos repetidos, es decir, dos vértices o aristas iguales en un mismo grafo (caso test)
 - La conversión de cada elemento de vértices y edges a mayúsculas para evitar el CaseSensitive

```

    /**
     * Constructor que inicializa el grafo con un conjunto de vértices y aristas.
     * Convierte todos los nombres de vértices a mayúsculas y elimina duplicados.
     *
     * @param vertices Lista de vértices
     * @param edges Matriz de aristas, donde cada arista es un par de vértices
     */

```

```

public class Graph {
    private ArrayList<String> vertices;
    private ArrayList<ArrayList<String>> edges;

    public Graph(String[] vertices, String[][] edges) {
        this.vertices = new ArrayList<>();

        for (String vertex : vertices) {
            String v = vertex.toUpperCase();
            if (!this.vertices.contains(v)) {
                this.vertices.add(v);
            }
        }

        for (String[] edge : edges) {
            for (int i = 0; i < edge.length; i++) {
                edge[i] = edge[i].toUpperCase();
            }
        }
    }

    this.edges = new ArrayList<>();
    for (String[] edge : edges) {
        if (edge.length == 2 && this.vertices.contains(edge[0]) && this.vertices.contains(edge[1])) {
            ArrayList<String> newEdge = new ArrayList<>(Arrays.asList(edge));
            if (!containsEdge(this.edges, newEdge)) {
                this.edges.add(newEdge);
            }
        }
    }
}

```

- Para el test ToString, se sobrescribió el método ToString de object de java.

```

    /**
     * Retorna una representación en cadena del grafo, se usa @Override ya que el método toString(),
     * se esta sobrescribiendo el método toString() de la clase base Object de java.
     *
     * @return Una cadena que representa las aristas del grafo en el formato (A, B) (C, D) ...
     */
    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder();

        for (int i = edges.size() - 1; i >= 0; i--) {
            sb.append("(").append(edges.get(i).get(index:0)).append(str:", ").append(edges.get(i).get(index:1)).append(str:") ");
            if (i < edges.size() - 1) {
                sb.append(str:" ");
            }
        }
        return sb.toString().strip();
    }
}

```

- Para el test shouldValidity se sobrescribió el método equals de java.

```

    /**
     * Verifica si dos grafos son iguales comparando vértices y aristas.
     */
    @Override
    public boolean equals(Object o) {
        if (!(o instanceof Graph)) {
            return false;
        }
        if ((this.vertices == ((Graph) o).vertices) && (this.edges == ((Graph) o).edges)) {
            return true;
        }
        return false;
    }
}

```

- También para que pasara la creación del grafo se hicieron los métodos que cuentan la cantidad de vértices y aristas.

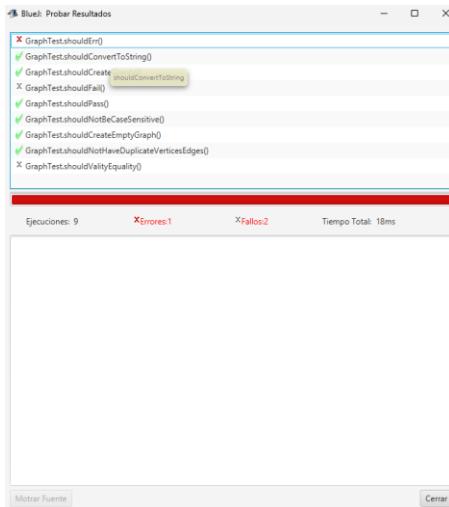
```

    /**
     * Obtiene la cantidad de vértices del grafo.
     *
     * @return Número de vértices
     */
    public int vertices() {
        return vertices.size();
    }

    /**
     * Obtiene la cantidad de aristas del grafo.
     *
     * @return Número de aristas
     */
    public int edges() {
        return edges.size();
    }

```

4. Capturas de las pruebas de la clase Graph.



Desarrollando GraphCalculator

BDD - MDD

[En lab02.doc, GraphCalculator.asta, *.java]

Para desarrollar esta aplicación vamos a considerar algunos ciclos. En cada 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
Piensen en los deberia y los noDeberia (should and shouldNot)
3. Diseñar los métodos
Usen diagramas de secuencia. En astah, creen el diagrama sobre el método correspondiente.
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 la calculadora: crear una calculadora y asignar y consultar un grafo

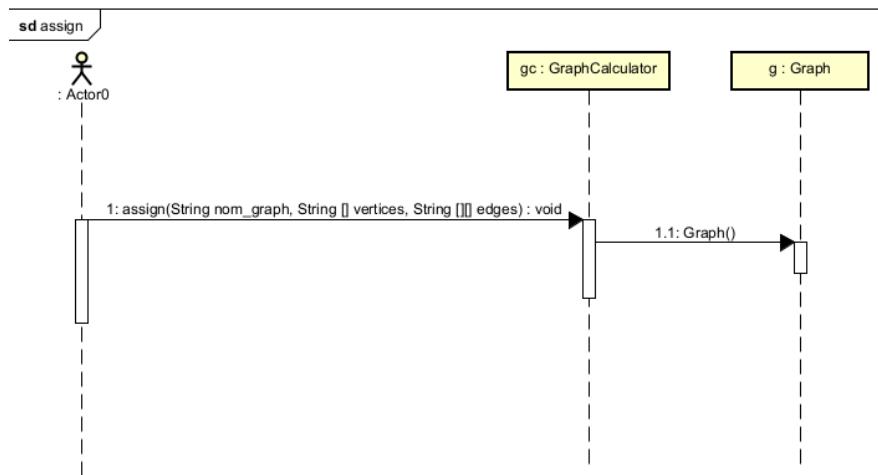
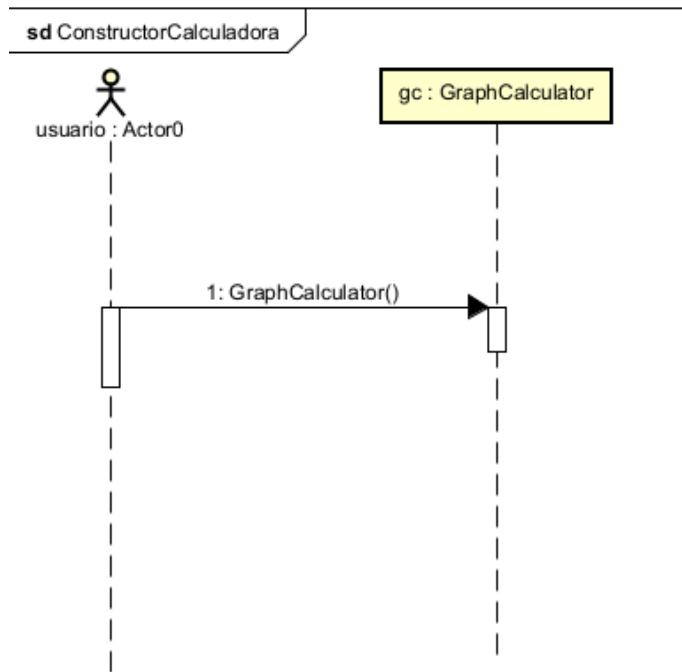
Ciclo 2 : Operaciones unarias: insertar y eliminar arcos; consultar si un conjunto de vértices pertenece al graph y retornar el camino que pasa por un conjunto de vértices

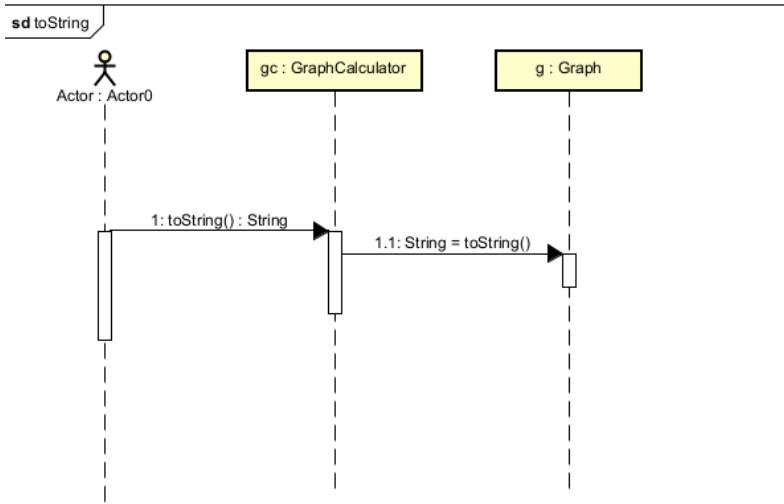
Ciclo 3 : Operaciones binarias: union, intersección, diferencia y junta

BONO Ciclo 4 : Defina dos nuevas operaciones

Ciclo 1

1. Los métodos base serían el constructor de la calculadora, asignaciónGrafo y toString.
2. Los casos de prueba serían:
 - ShouldCreateCalculator
 - ShouldAssign
 - ShouldConvertToString
3. Diseño de métodos





4.

```


/**
 * Constructor de calculadora de grafos
 */
public GraphCalculator() {
    this.variables = new TreeMap<String, Graph>();
    ultGrafo = null;
    status = false;
}

/**
 * Assign a graph to an existing variable
 * a := graph
 *
 * @param : String, String [], String[][]
 */
public void assign(String nombre_graph, String[] vertices, String[][] edges) {
    Graph grafo = new Graph(vertices, edges);

    variables.put(nombre_graph, grafo);

    status = true;
}


```

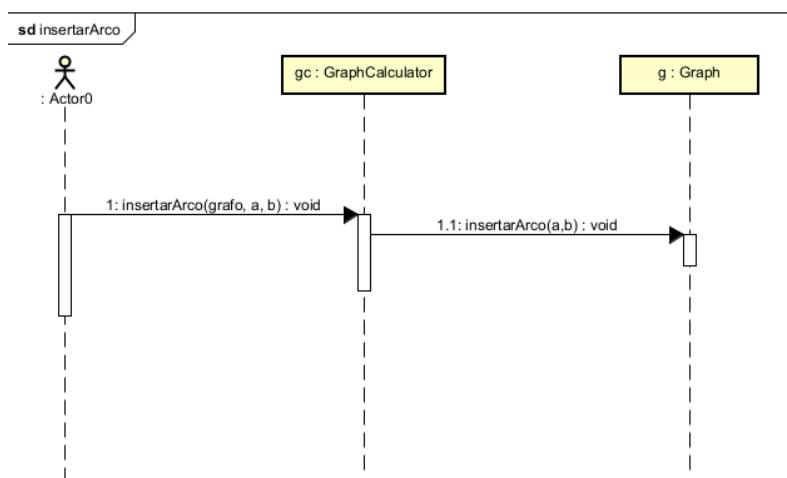
```

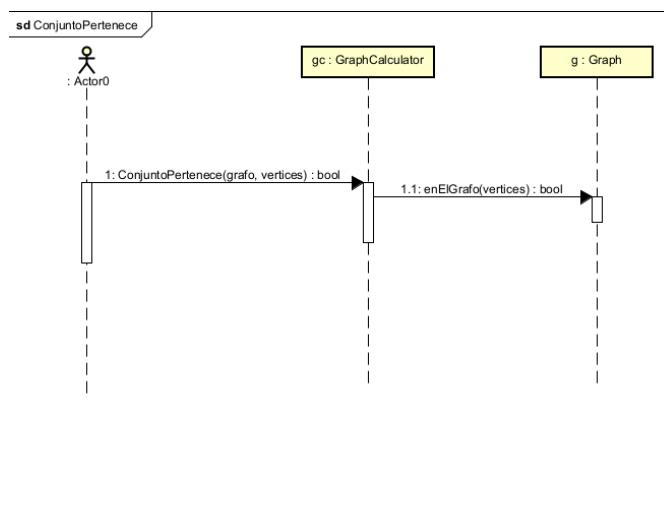
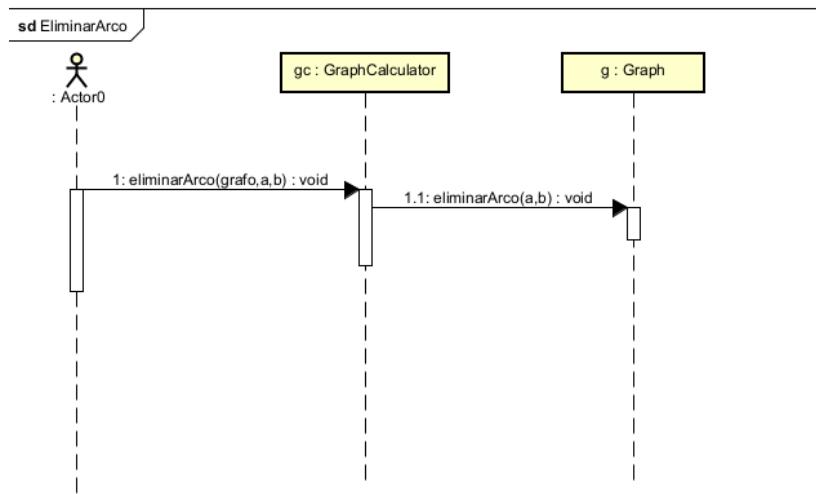
/**
 * Returns the graph with the edges in uppercase in alphabetical order.
 */
public String toString(Graph graph) {
    return graph.toString();
}

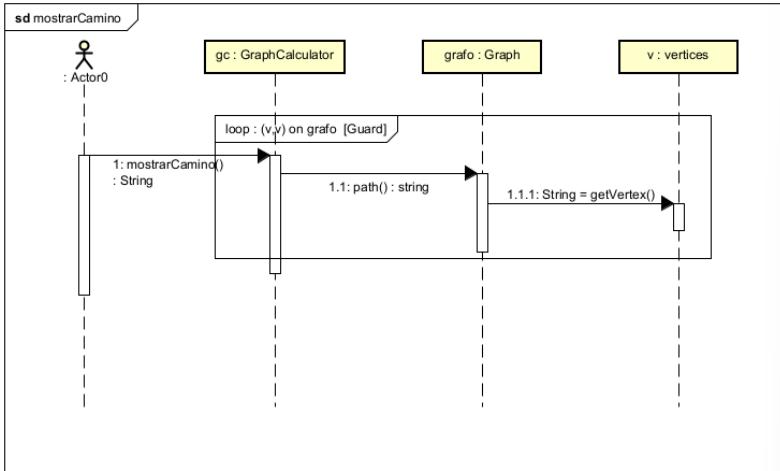
```

CICLO 2

1. Los métodos base de este mini-ciclo son insertarArco, eliminarArco, conjuntoPertenece, mostrarCamino.
2. Los casos de prueba correspondientes son:
 - ShouldinsertarArco
 - ShouldeliminarArco
 - ShouldconjuntoPertenece
 - ShouldmostrarCamino (no se llevó a cabo)
3. Diseño de métodos







4.

```

/*
 * Función que inserta un arco entre dos vertices al conjunto de arcos
 * del grafo dado.
 * @param Graph grafo, String vertice a , String vertice b
 * @return void
 */
private void insertarArco(Graph grafo, String a, String b) {
    grafo.insertarArco(a, b);
}

/*
 * Función que elimina un arco entre dos vertices del conjunto de arcos
 * del grafo dado.
 * @param Graph grafo, String vertice a , String vertice b
 * @return void
 */
private void eliminarArco(Graph grafo, String a, String b) {
    grafo.eliminarArco(a, b);
}

```

```

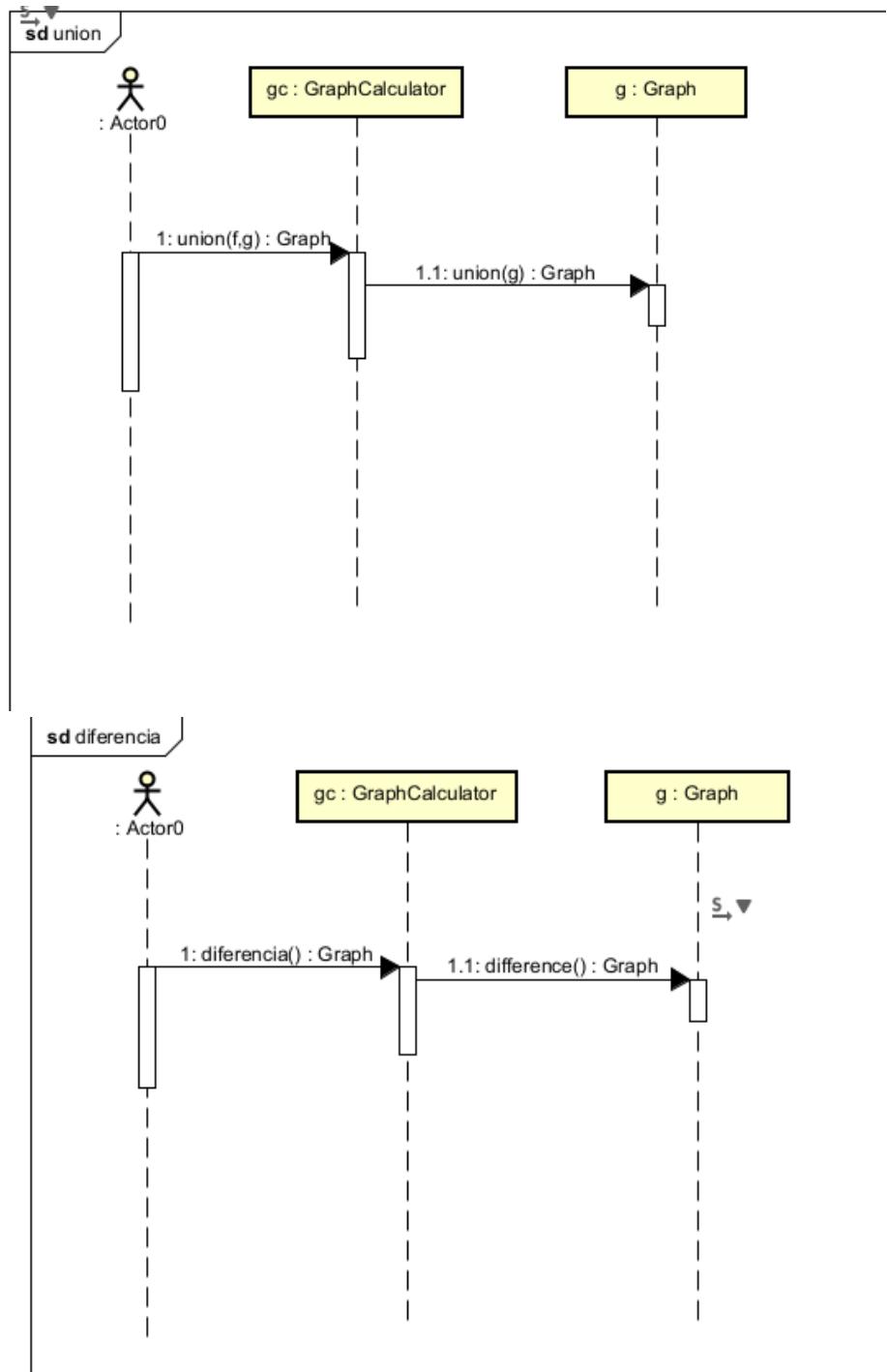
private boolean conjuntoPertenece(Graph grafo, String[] vertices) {
    ArrayList<String> lVertices = new ArrayList<>(Arrays.asList(vertices));

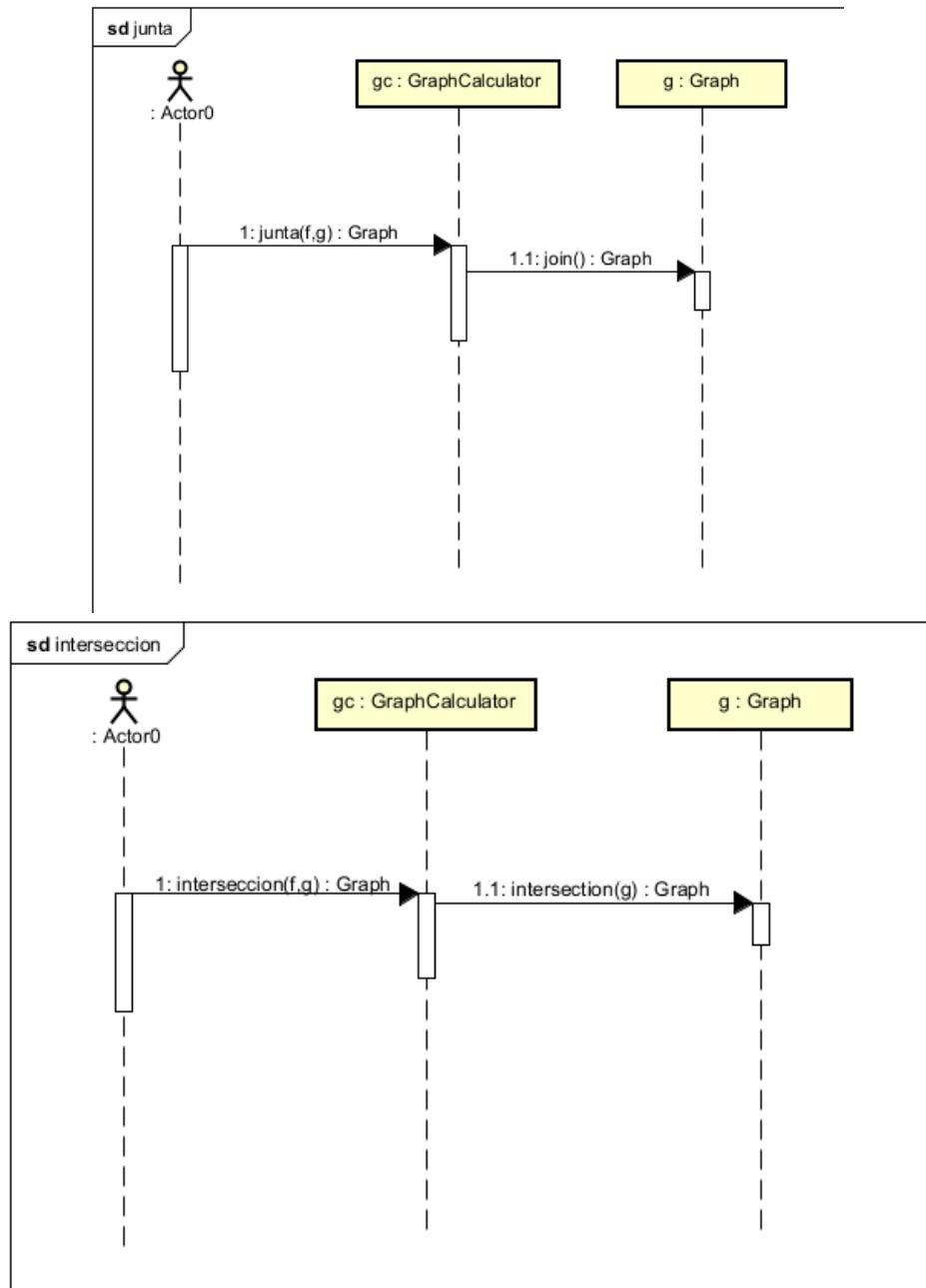
    for (int i = 0; i < lVertices.size(); i++) {
        String vertice = lVertices.get(i);
        boolean booleano = grafo.enElGrafo(vertice);
        if (booleano == false) {
            return false;
        }
    }
    return true;
}

```

CICLO 3

1. Los métodos base serían unión , intersección ,diferencia , junta y un método que se encargue de asignar el valor de la operación a una variable.
2. Los casos de prueba son :
 - o ShouldUnion
 - o Shouldintersection
 - o ShouldJoin
 - o ShouldDifference
3. Diseño de métodos





4.

```

public Graph assignBinary(String a, String b, char op, String c){

    Graph bgrafo = variables.get(b);
    Graph cgrafo = variables.get(c);
    Graph agrafo = null;

    switch (op) {

        case 'u':
            agrafo = bgrafo.union(cgrafo);
            break;

        case 'i':
            agrafo = bgrafo.intersection(cgrafo);
            break;

        case 'd':
            agrafo = bgrafo.difference(cgrafo);
            break;

        case 'j':
            agrafo = bgrafo.join(cgrafo);
            break;
    }
    return agrafo;
}

```

Punto 5

Ciclo	Graph Calculator	GraphCalculatorTest
1	GraphCalculator, assign, toString	ShouldCreateCalculator,ShouldAssign,shouldConvertToString
2	assignUnary	ShouldInsertarArco, shouldEliminarArco, ShouldConjuntoPertenece
3	assignBinary	shouldUnion, shouldIntersection, ShouldJoin, shouldDifference
4	N/A	N/A

REFERENCIAS

Vidal, S. (2024, 13 enero). Cómo abrir un archivo CLASS ➤ . *Tecnobits*.
<https://tecnobits.com/como-abrir-un-archivo-class/>

CTXT file extension. (2013, 2 octubre). <https://fileinfo.com/extension ctxt>

Vector (java platform SE 8). (2024, diciembre 4). Oracle.com.
<https://docs.oracle.com/javase/8/docs/api/java/util/Vector.html>

<https://docs.oracle.com/javase/8/docs/api/java/util/TreeMap.html>

ArrayList (java platform SE 8). (2024, diciembre 4). Oracle.com.
<https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html>

GeeksforGeeks. (2024, octubre 17). *Difference Between Failure and Error in JUnit*.

GeeksforGeeks. <https://www.geeksforgeeks.org/difference-between-failure-and-error-in-junit/>

tutoriales.duhnnae.com. (s. f.). *Aprender programación - Sencillos ejercicios*. Duhnnae.

https://tutoriales.duhnnae.com/tutorial_hacer_menu_switch_case_en_java_programacion_basica.php

Converting 'ArrayList to «String[]» in Java. (s. f.-b). Stack Overflow.

<https://stackoverflow.com/questions/4042434/converting-arrayliststring-to-string-in-java>

Retrospectiva

Tiempo total invertido por cada integrante,

Carolina: 19 h

Juanita: 18h30min

¿Cuál es el estado actual del laboratorio? y ¿por qué?

- Incompleto, por mi parte en la clase Grafo faltó la implementación de los métodos Path y buscar vecinos, por falta de tiempo y confusión en como realizar la implementación.

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

- La práctica de TDD, ya que el tener las pruebas antes de ir a codificar nos permitió guiarnos sobre lo que necesitábamos tener como parámetros del método y lo que se debía retornar.

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

- Por mi parte, lograr hacer el cambio de tipos y poderlo manejar de una manera adecuada, además, de hacer una correcta implementación de lo que es la sobrecarga en constructor y de programación funcional para el método Unión grafo.
- Por mi parte, entender el cambio de tipos, ya que, nos encontramos con muchos tipos de variables (ArrayList, ArrayList de ArrayList, etc), y llevar este entendimiento a la hora de plantear las pruebas.

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

- El manejo de tipos, hacer los cambios por medio de copia por referencia y el uso de ArrayList.
- El manejo de equals a la hora de hacer los tests de GraphCalculator ya que, a pesar de que los resultados son correctos, equals compara las referencias de los objetos y no el contenido.

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

- La distribución del trabajo.
- La comunicación. A pesar de que durante este laboratorio mejoramos mucho en este tema y en la distribución del tiempo, considero que podemos progresar más en este aspecto.

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

Se presentan las referencias en la parte de Bibliografía, de las más útiles fue *Converting 'ArrayList to «String[]» in Java* ya que permitió llevar a un mejor entendimiento de los tipos de variables usados durante el laboratorio.