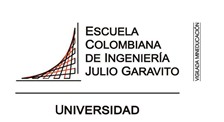
****

**Escuela Colombiana de Ingeniería Julio Garavito**

**Programación Orientada a Objetos 2024-2**

**Laboratorio II**

**Cristian Santiago Pedraza Rodríguez**

**Andersson David Sánchez Méndez**

**6 de septiembre de 2024**

**ESCUELA COLOMBIANA DE INGENIERÍA PROGRAMACIÓN ORIENTADA A OBJETOS**

**Diseño y Pruebas. Interacción entre objetos. 2024-2**

**Laboratorio 2/6**

## 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](http://www.extremeprogramming.org/rules/crccards.html) for design sessions. **Testing** All code must have [unit tests](http://www.extremeprogramming.org/rules/unittests.html).

# 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.
  + En las entregas deben indicar el estado de avance de su laboratorio y los problemas pendientes por resolver.
  + 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**

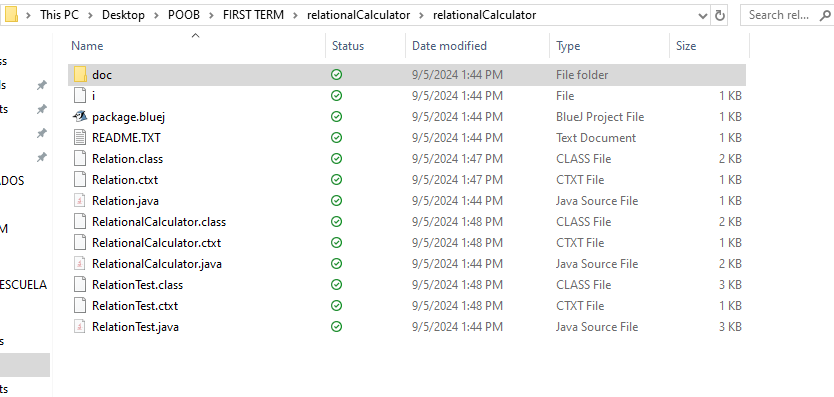
**Objetivo**

En matemáticas una **relación** es una propiedad que asigna un valor de verdad a combinaciones de k datos (k-tuplas). Las relaciones se representan en informática como tablas bidimensionales: las columnas son los atributos de la relación y las filas son las tuplas (los datos almacenados).

El objetivo de este laboratorio es implementar una calculadora para operar relaciones.

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

1. El proyecto “relationalCalculator” 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.



Al revisar el directorio, nos dimos cuenta de que había un total de 13 elementos, 1 folder y 12 archivos con diferentes tipos de extensión.

Las extensiones son:

.bluej, .txt, .class, ctxt y .java.

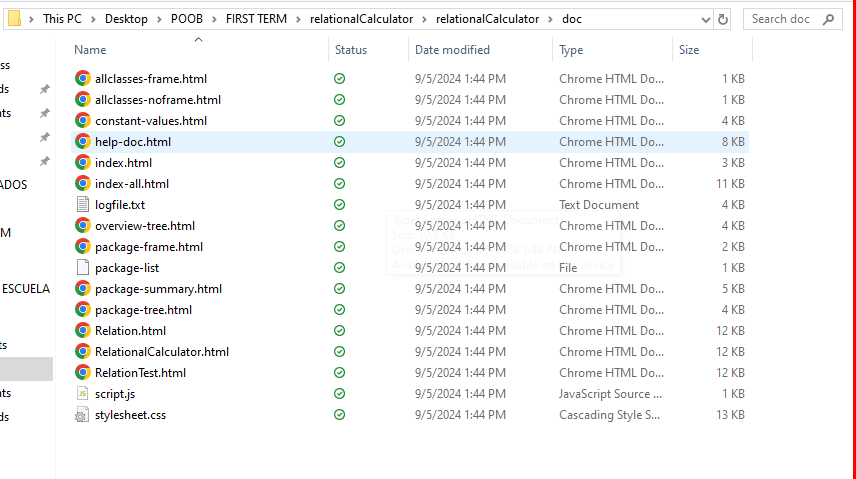
.bluej contiene información sobre la estructura del proyecto, como las clases incluidas, sus relaciones y configuraciones de BlueJ,

.txt contiene el README,

.class contiene el bytecode generado por el compilador de Java a partir de un archivo .java,

.ctxt contiene información contextual para el proyecto, aunque no es una extensión estándar,

.java contiene el código fuente escrito en Java. Cuando se compila, se genera el archivo .class que luego puede ser ejecutado por la JVM.

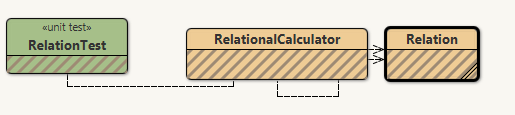


Y en el folder llamado “doc” está la documentación de todas las clases que están el BlueJ(html), así como archivos de estilo css y javascript.

1. Exploren el proyecto en BlueJ

¿Cuántas clases tiene?

Tiene 3 clases.



¿Cuál es la relación entre ellas?

Las clases RelationTest y RelationalCalculator usan la clase Relation.

¿Cuál es la clase principal de la aplicación?

La clase principal de la aplicación es RelationalCalculator

¿Cómo la reconocen?

***Porque es la que ofrece todos los servicios y además es la clase de pruebas.***

¿Cuáles son las clases “diferentes”?

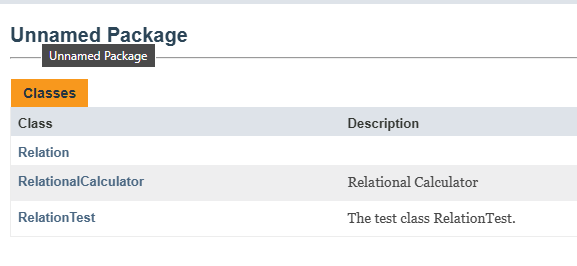
RelationTest

¿Cuál es su propósito?

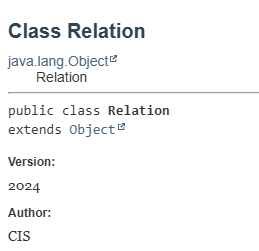
Su propósito es hacer pruebas unitarias, o pruebas a partes específicas del código Relation. Esto quiere decir, pruebas para garantizar el correcto funcionamiento de los comportamientos de dicha clase.

Para las siguientes dos preguntas sólo consideren las clases “**normales**”:

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

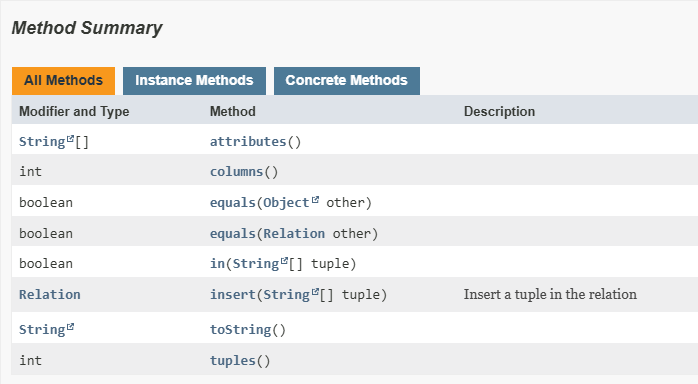


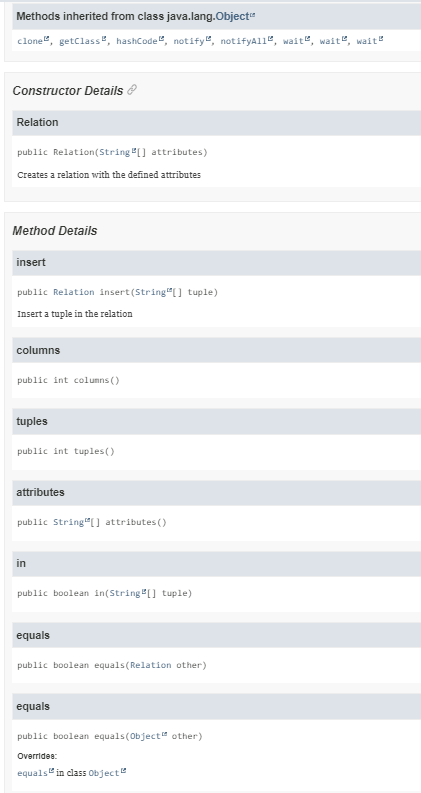
Relation

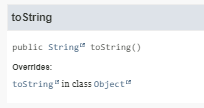


A la clase Relation le hace falta su descripción.



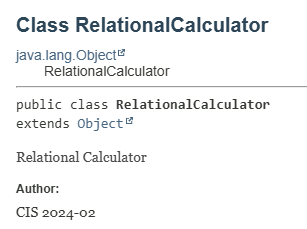


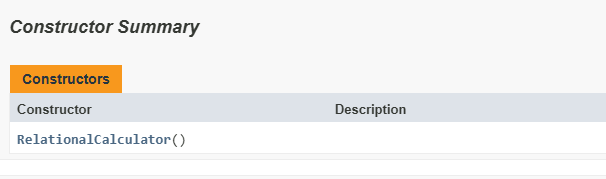




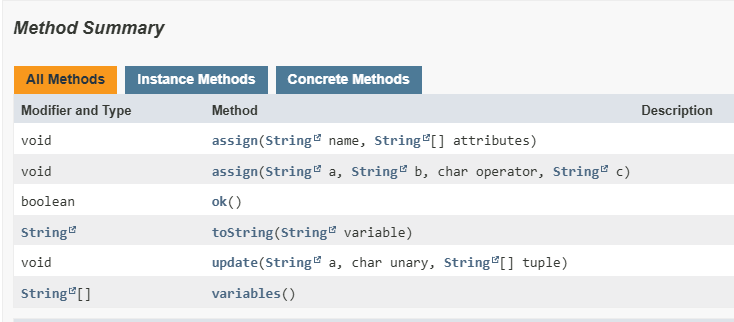
Así como varias descripciones de varios métodos no tienen descripción(parámetros).

RelationalCalculator





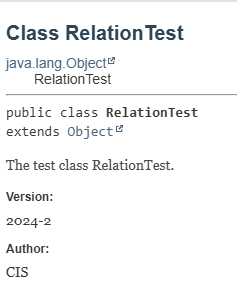
El constructor no tiene una descripción.

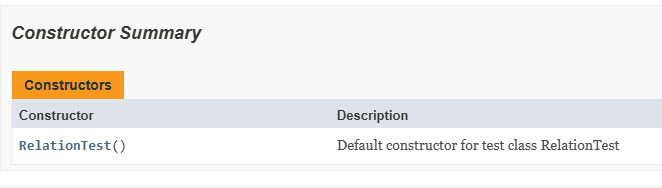




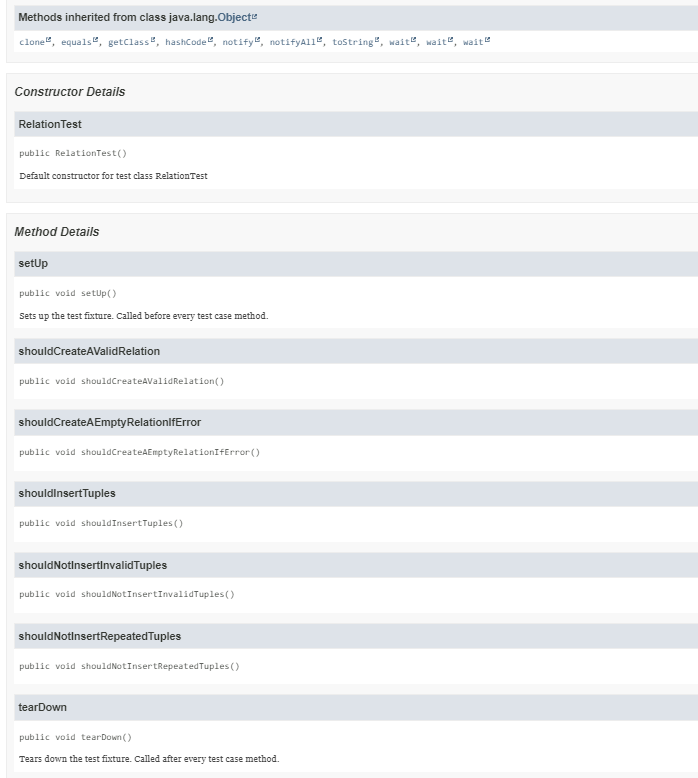
Ninguno de los métodos tiene descripción, así como también los parámetros que reciben tampoco la tienen.

RelationTest









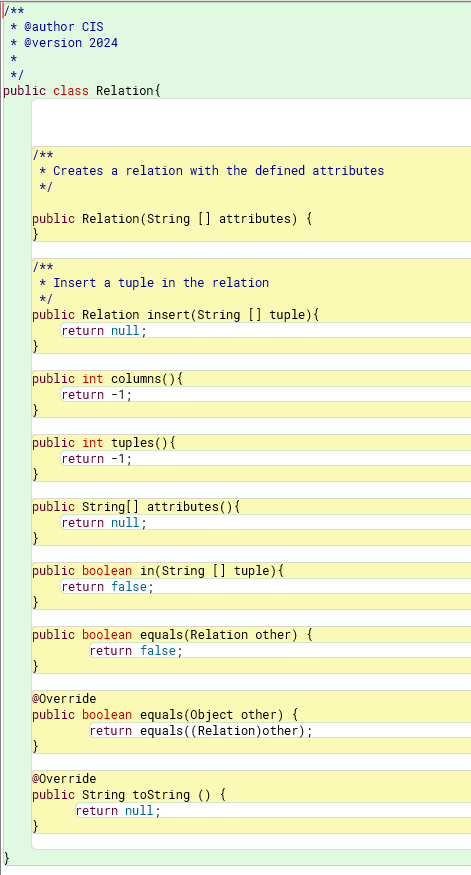
Salvo dos métodos, todos los demás no tienen descripción.

1. 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)
2. Relation
   1. Atributos y métodos

Los atributos no están creados en el constructor, solo está una instancia de la clase misma. Además, algunos métodos están incompletos o le faltan comentarios para añadir algo a los métodos.

* 1. Código, documentación y comentarios

Así como se vio en la documentación, el mismo código genera las mismas faltas, es decir, en algunos métodos faltan las descripciones o comentar lo que hace cada método o también decir la función que cumple cada parámetro dentro del método.

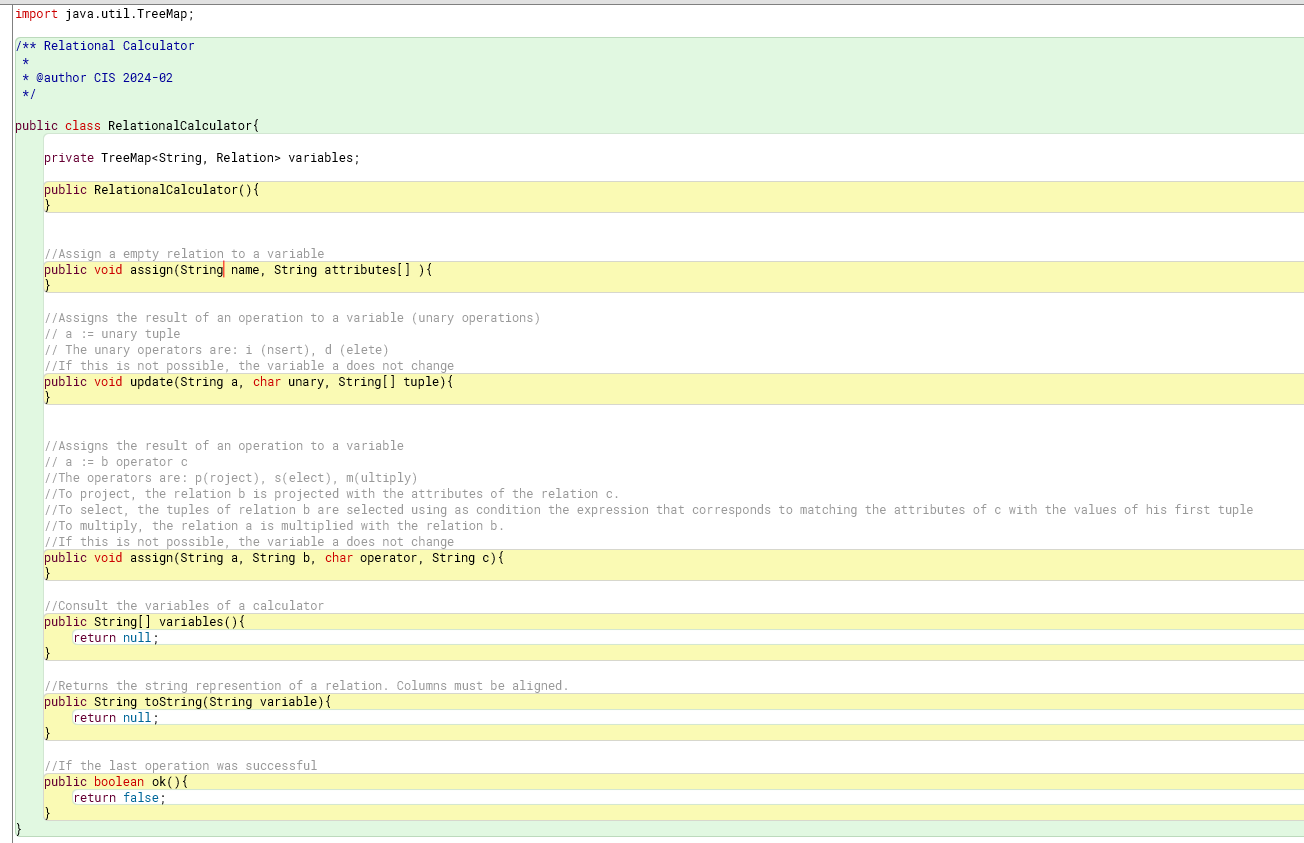


1. RelationalCalculator
   1. Atributos y métodos

Algunos métodos retornan null, pero la lógica del programa debería retornar algo más robusto, dependiendo de los comentarios que aparecen en el código; y otros están incompletos.

* 1. Código, documentación y comentarios

Vemos que el código está comentado acerca de la función que tienen que hacer los métodos, pero no están documentados los parámetros de los métodos como operator. El código está incompleto.

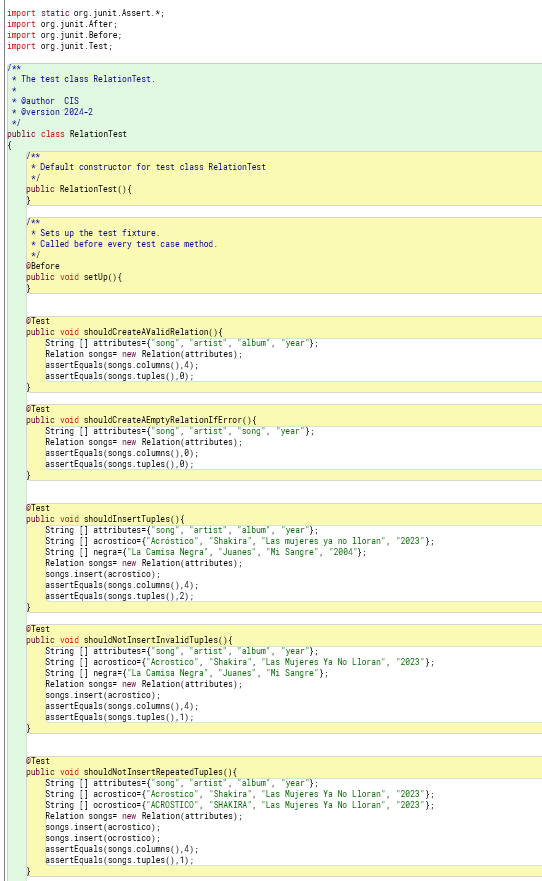


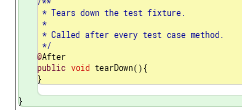
1. RelationTest
   1. Atributos y métodos

Vemos que el constructor no tiene nada de contenido, solo está como comentario, pero no tiene la lógica del programa. Algunos métodos en esta clase si cumplen un rol importante en la clase, mientras que otros están incompletos.

* 1. Código, documentación y comentarios

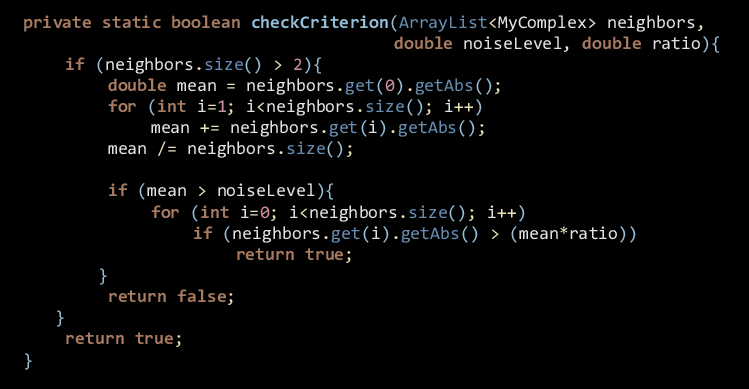
Algunos métodos sí tienen la lógica del programa, sin embargo no están comentados acerca de lo que hacen.



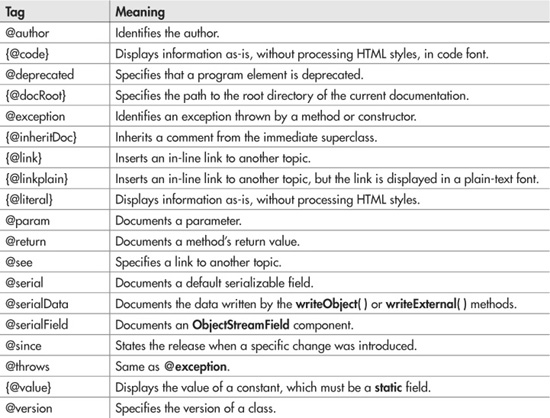


¿Qué diferencia hay entre el código, la documentación y los comentarios?

El código se basa en la sintaxis del lenguaje de programación Java, así es un conjunto de líneas de texto que expresan, en cualquier lenguaje de programación, los pasos que debe seguir el computador(máquina) para la correcta ejecución de un programa específico.

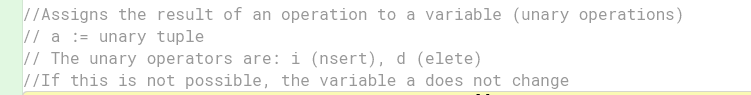


Documentar sirve para añadir suficiente información para explicar lo que hace el programa en general, se hace uso de este cuando se genera la respectiva documentación de la clase, aquí se muestra la descripción del constructor, métodos (con parámetros), y las APIs si se usan. También la manera en que se hace esto en Java es con la etiqueta(símbolo) @, indicando si es un parámetro o si retorna algo o simplemente para dar una descripción del método.



REFERENCIA: [DocumentTags](https://www.oreilly.com/library/view/java-a-beginners/9780071606325/appblev1sec1.html#:~:text=Document%20tags%20that%20begin%20with,used%20within%20a%20larger%20description).

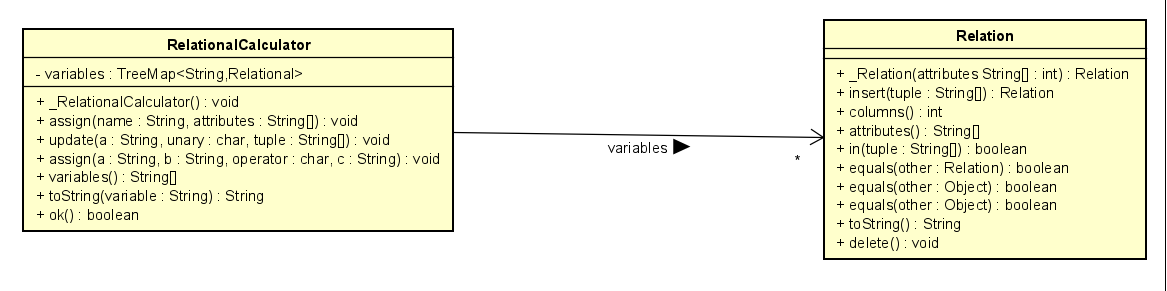
Comentar es dentro del mismo código, escribir pequeñas descripciones acerca del paso a paso por instrucciones, que será útil cuando se pase el código a otra persona y que pueda entender cada método, o en sí lo que hace la clase o todo el programa en general.



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

# MDD MODEL DRIVEN DEVELOPMENT

1. Realice el diagrama de clases correspondiente al proyecto. (No incluya la clase de pruebas)



1. ¿Cuáles contenedores están definidos?

El único contenedor que existe está definido en la clase RelationalCalculator y es:



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

Consulte la API de java.

Según la documentación de Java. Vamos a abordar cada una de las clases.

1. **ArrayList**

La documentación nos dice que nos permite una implementación redimensionable de la interfaz List, implementando todas las operaciones de las listas y permite todos los elementos, incluido el *null.* También menciona que es muy parecida a **Vector,** excepto por el hecho de que no está sincronizada.

1. **Vector:**

La clase Vector implementa una variedad creciente de objetos, como las matrices, contiene componentes que pueden ser accedidos usando enteros como índices. Sin embargo, el tamaño del vector puede ser modificado para crecer o disminuir para acomodarse a adiciones y remociones de elementos después de que el vector ha sido creado. Esto se debe a que el vector intenta optimizar el manejo de memoria. La documentación nos invita que, si una implementación segura de subprocesos no es necesitada, mejor utilizar una ArrayList

1. **TreeMap:**

Por otro lado, y siendo muy interesante, es que la clase TreeMap es una implementación de NavigableMap basada en un árbol rojo-negro. El mapa es organizado de acuerdo al orden natural de sus llaves o por un comparador suministrado al momento de la creación del mapa, dependiendo del constructor usado.

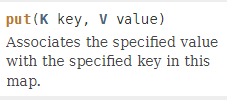
Esta implementación garantiza un tiempo de log(n) para las consultas y las operaciones (constainkey, get, put y remome). Lo más interesante es que los algoritmos son adaptaciones de los que están en Cormen en introducción a los algoritmos.

Entonces **la diferencia principal es** ArrayList es para el manejo dinámico de una colección de objetos, mientras que un vector no es tan dinámico, pero es mejor para subprocesos por su naturaleza sincrónico, mientras que la implementación de treeMap nos da una poderosa estructura de datos para hacer consultas y operaciones en tiempos lg(n) debido a su política de ordenamiento de datos.

1. En el nuevo contenedor,

¿Cómo adicionamos un elemento?

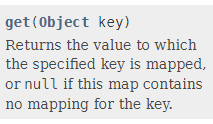
Para adicionar un nuevo elemento, utilizamos el método put, en donde enviamos como argumentos el valor y la llave.



Para adicionar un nuevo elemento en el contenedor TreeMap

¿Cómo lo consultamos?

Para consultar un elemento en nuestro árbol, utilizamos el método get y le enviaremos como argumento la llave el elemento:



¿Cómo lo eliminamos?

Y para eliminarlo utilizamos el método remove, en donde pasamos como argumento la llave con la que está asociado el objeto que deseamos eliminar:



**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 RelationTest

1. Revisen el código de la clase RelationTest.

¿cuáles etiquetas tiene (componentes con símbolo @)?

Hay varias etiquetas con ese símbolo:

* Partiendo de la documentación principal del código donde está @author y @version que se refieren al nombre de quien crea el código y la versión de Java en el que se está trabajando.
* Existe otro símbolo @Before que se ejecuta o se llama antes de cada caso de prueba.
* Los siguientes símbolos se refieren a los de prueba @Test (hay 5).
* Otro símbolo @After llamado después de cada caso de prueba.
* El último símbolo @Override debería estar sobreescribiendo un método de alguna clase padre. Si no es así, el compilador de java generará un error.

¿cuántos métodos tiene?

Tiene 7 métodos

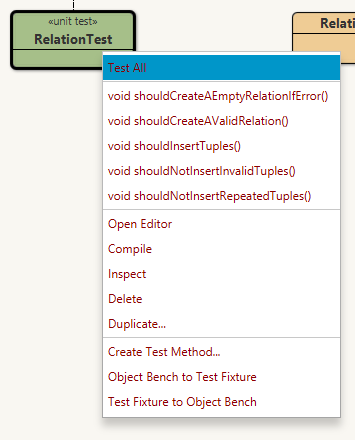
¿cuántos métodos son de prueba?

Hay 5 métodos que son de prueba

¿cómo los reconocen?

Se reconocen por la etiqueta del símbolo @Test encima de cada método.

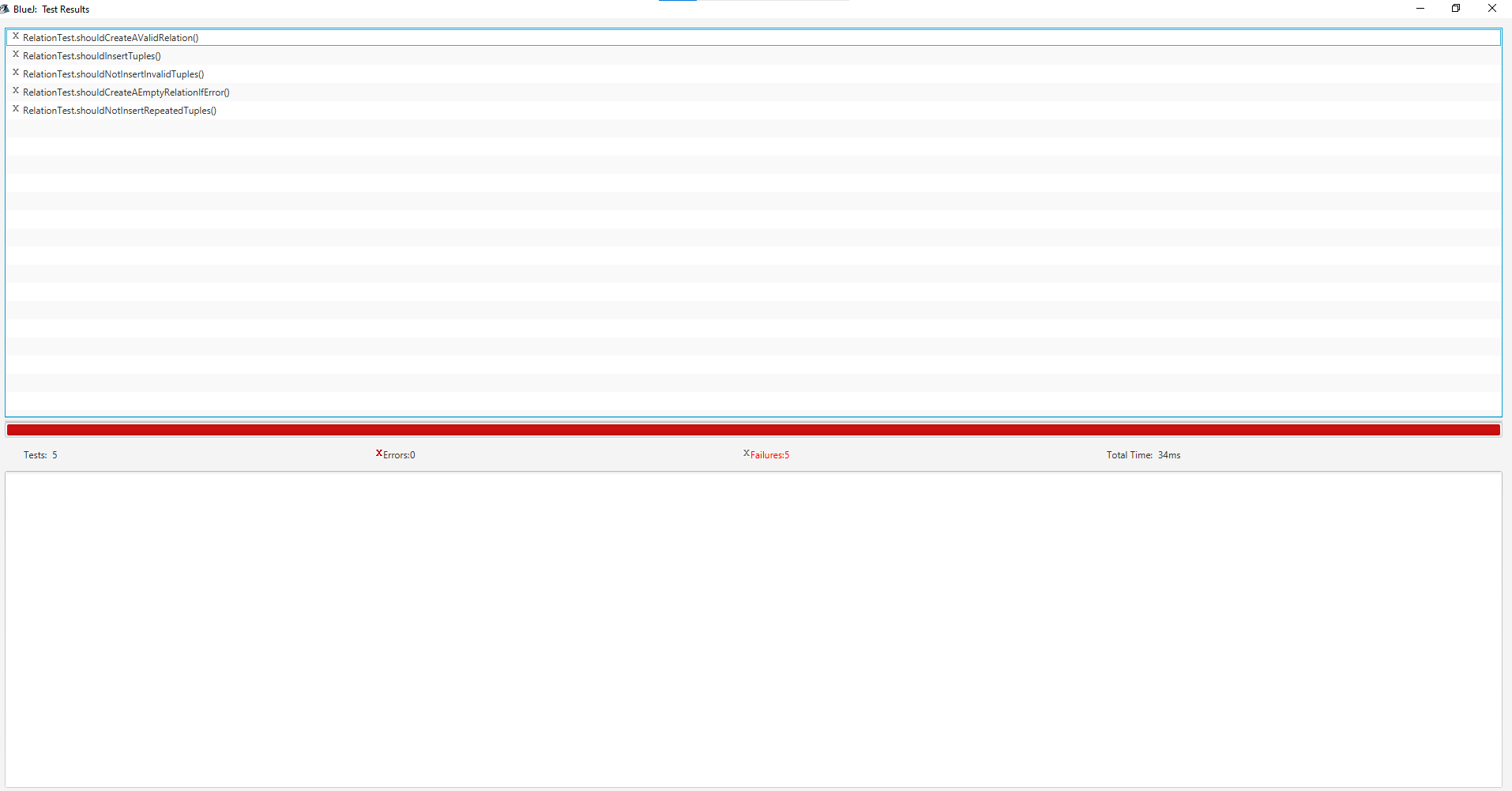
1. Ejecuten los tests de la clase RelationTest. (click derecho sobre la clase, Test All)



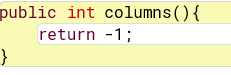
¿Cuántas pruebas se ejecutan?

Se ejecutan 5 pruebas.

¿Cuántas pasan? ¿por qué? Capturen la pantalla.

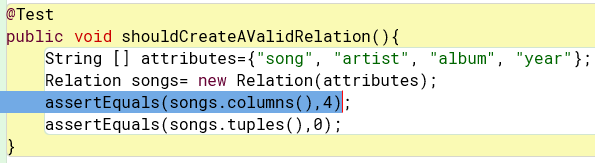


Ninguna pasa, ya que para cada prueba genera un error común en assertEquals(song.columns(),4) o assertEquals(song.columns(),0) porque está relacionado a cómo la clase Relation está implementada en el método columns, cómo está actualmente, siempre el método columns va a retornar -1, por lo tanto, assertEquals no va a retornar el número correcto de columns y siempre va a fallar.



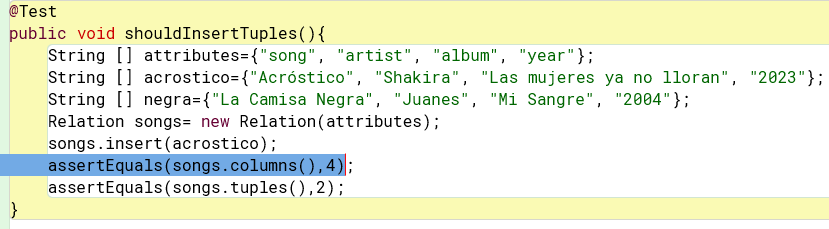
**RelationTest.shouldCreateAValidRelation()**





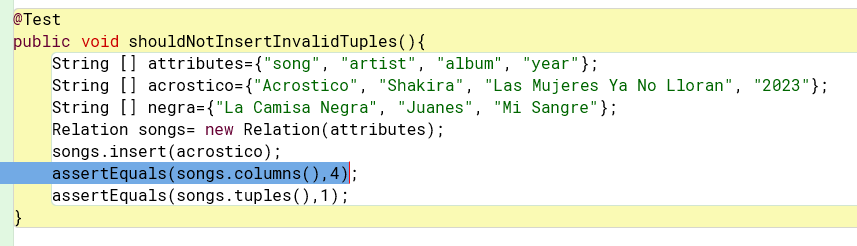
**RelationTest.shouldInsertTuples()**



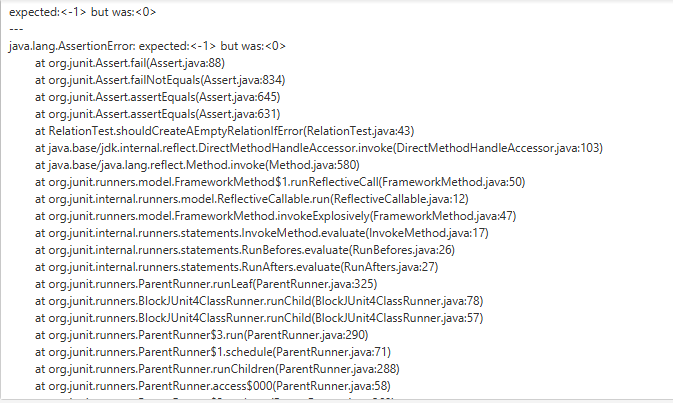


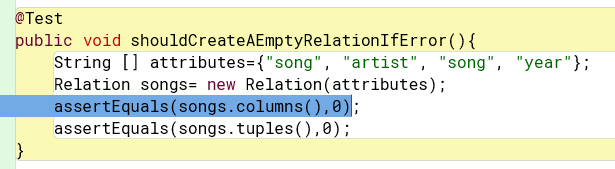
**RelationTest.shouldNotInsertInvalidTuples()**



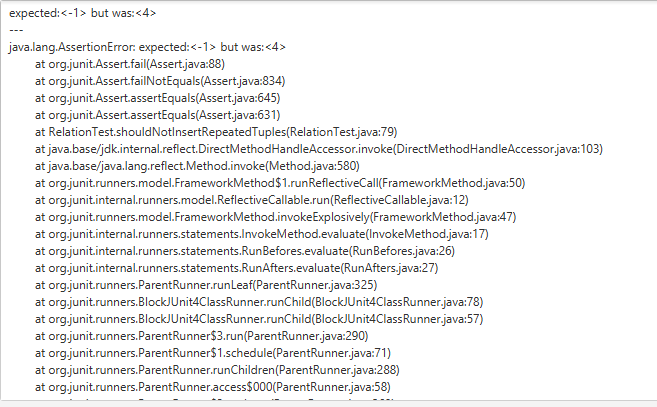


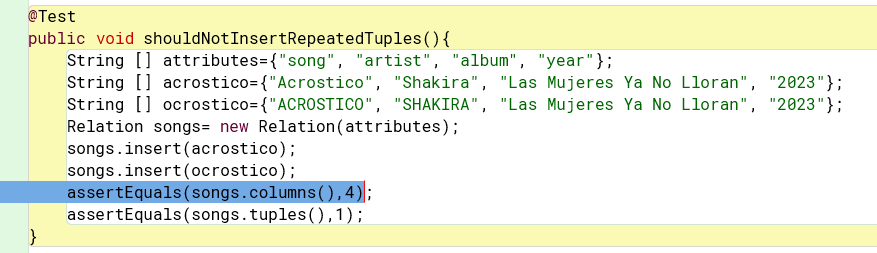
**RelationTest.shouldCreateAEmptyRelationIfError()**





**RelationTest.shouldNotInsertRepeatedTuples()**





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

@author: Indica quien escribió o es responsable del código, su uso es para mostrar el nombre del autor de la clase o el archivo del código, en este caso CIS es el autor de RelationTest.

@version: Proporciona la versión del código, es útil para el control de versiones y la documentación, en este caso está en la versión 2024-2.

@Before: Indica un método que debe ejecutarse antes de cada prueba en una clase de test, se usa para la inicialización de variables o configuración de objetos, en este caso setUp() hace esta función.

@Test: Indica que el método es una prueba unitaria, se usa para verificar el correcto comportamiento del código, en este caso hay 5 métodos que JUnit los ejecutará como pruebas.

@After: Indica un método que debe ejecutarse después de cada prueba en una clase de test, se usa para limpiar o resetear el entorno después de la ejecución de tests, es decir, reinicia variables o restaura el estado original, en este caso el método tearDown() cumple el rol de esa función.

@Override: Se utiliza para forzar al compilador a comprobar o detectar en tiempo de compilación que, si se está sobrescribiendo correctamente un método, para así evitar errores en tiempo de ejecución, los cuales serían mucho más difíciles de detectar sin esa documentación.

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

Recurriendo a la clase Assert del API Junit para definir en nuestras propias palabras:

assertTrue: El propósito se basa en verificar que una condición booleana sea verdadera, si el valor es true el método se comprueba o sino la prueba fallará. Se puede acompañar de un mensaje, además de mostrar el resultado del método.

assertTrue(“El valor esperado debe ser verdadero”,5>3);

assertFalse: Con la misma lógica de assertTrue, este método tiene como propósito se basa en verificar que una condición booleana sea falsa, si el valor es false el método se comprueba o sino la prueba fallará. Se puede acompañar de un mensaje, además de mostrar el resultado del método.

assertFalse(“El valor esperado debe ser falso”,5<3);

assertEquals: El propósito se basa en verificar si dos valores son iguales, si dos objetos o valores son iguales,se verifica o sino la prueba fallará. Se puede acompañar de un mensaje, además de mostrar el resultado del método.

assertEquals(“El valor esperado debe ser verdadero”,9,9);

assertArrayEquals: El propósito se basa en verificar si dos arreglos son iguales,comparar dos arrays para asegurarse que tiene los mismos elementos en el mismo orden, con esto se verifica o sino la prueba fallará. Se puede acompañar de un mensaje, además de mostrar el resultado del método.

assertArrayEquals(“El valor esperado debe ser falso”,{1,2,4},{1,2,3});

assertNull:El propósito se basa en verificar si un objeto es nulo, si lo es, se verifica o sino la prueba fallará. Se puede acompañar de un mensaje, además de mostrar el resultado del método.

assertNull(“El valor esperado debe ser verdadero”,null));

fail: El propósito se basa en hacer que la prueba falle de manera intencional, se puede marcar una prueba como fallida cuando se quiere asegurar de que una cierta parte del código no debe ejecutarse. Se puede acompañar de un mensaje, además de mostrar el resultado del método. (**try-catch)**

fail(“Esta prueba fallará intencionalmente”);

1. 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, shouldErr.

**FALLOS**

Ocurre cuando una aserción (como assertEquals, assertTrue, etc.) en una prueba no se cumple.

Indican que el comportamiento del código no coincide con el resultado esperado en la prueba. Significan que el test se ejecutó correctamente, pero el código probado no produjo los resultados deseados o requeridos.

@Test

public void testSuma() {

int resultado = 2 + 2;

assertEquals(5, resultado);

}

En este ejemplo vemos que la expectativa (lo que se quiere llegar es 5), pero al hacer la suma da 4, entonces habría un fallo, porque no se generó lo esperado.

Para añadir, el test reporta un fallo, pero no se generan mensajes de excepción porque los fallos representan problemas lógicos en el código o en las suposiciones de la prueba.

**ERROR**

Ocurre en la ejecución del código y vemos que lanza una excepción inesperada durante ese proceso, lo que interrumpe la prueba(en BlueJ cuando pasa esto, señala la línea de código que genera el error, o en otros casos, se crea un panel mostrando el respectivo error).

Estos suelen ocurrir debido a problemas técnicos en el código, como una excepción no manejada (por ejemplo, DivideByZero,acceder a elementos que no están en el arreglo, instanciar algo que no se ha creado, etc).

@Test

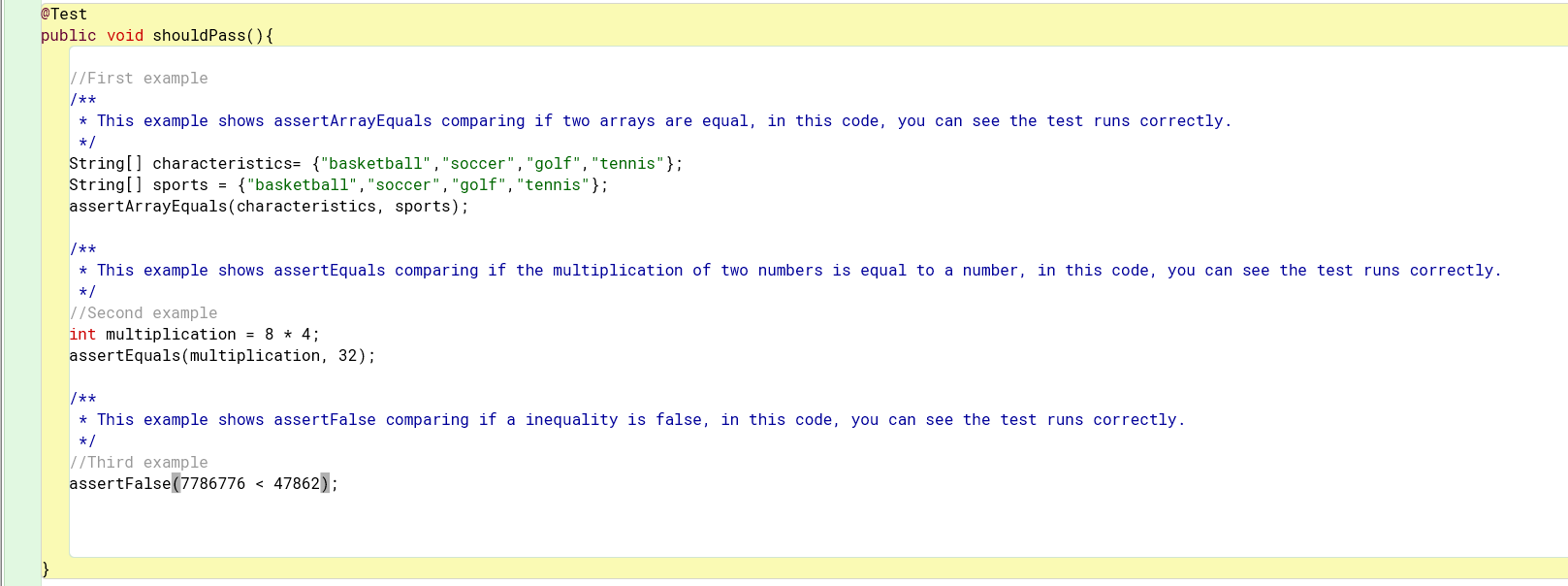
public void testDivision() {

int resultado = 10 / 0;

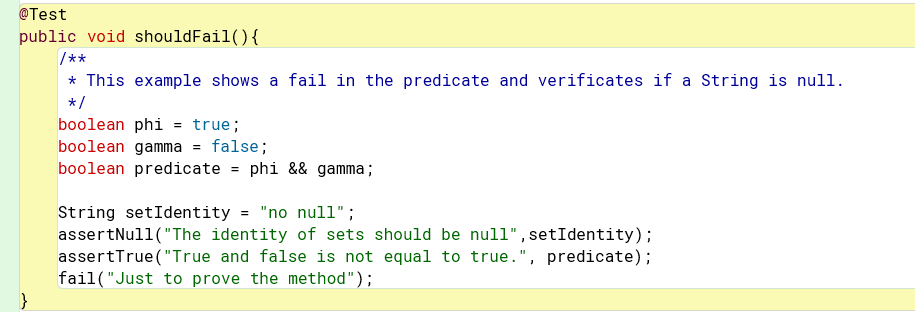
}

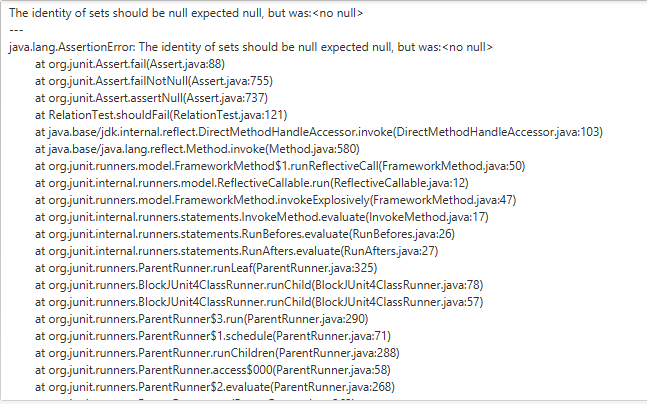
En este caso, se produce un ArithmeticException porque se intenta dividir entre cero, lo que lanza una excepción durante la prueba. El error no está relacionado con la verificación de la lógica de la prueba, sino con un problema en la ejecución del código(faltas de verificación en algunos métodos).

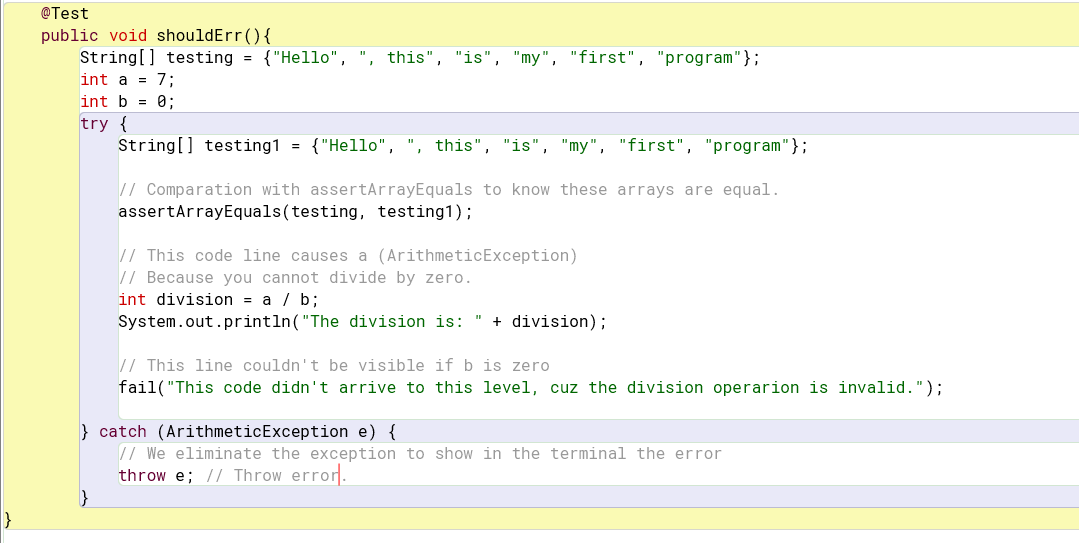
Tener errores en un código hace que no se puedan hacer pruebas unitarias porque no se va a ejecutar correctamente.



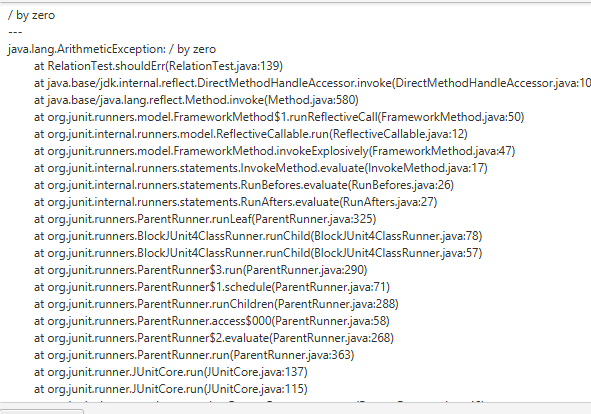










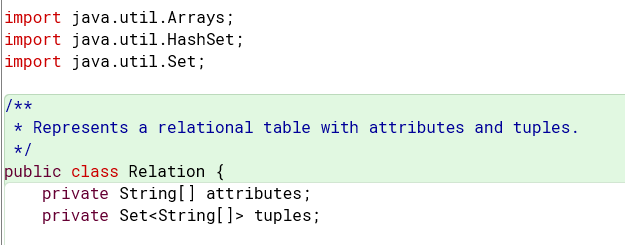


**Practicando Pruebas en BlueJ** [En lab02.doc \*.java]

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

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

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



Se pusieron estos atributos porque son los fundamentales para crear la calculadora relacional, uno es los atributos, que como su nombre lo indica es donde van a quedar guardados los valores que adopten(tuplas).

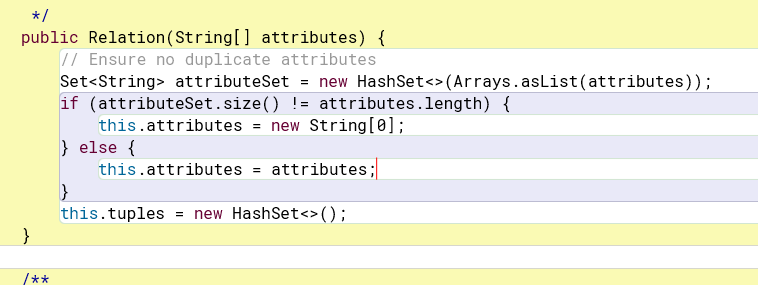
1. Determinen el invariante de la clase Relation. Justifique la decisión.

Respecto a los atributos de clase Relation, el invariante(relación en el estado de las variables que nunca cambia al inicio, durante y final del programa) son estos mismos porque de acuerdo a estos es que se hacen efectivos los métodos y la creación de las tuplas. También un invariante podría ser el nombre que se asigna a la relación en el método assign, porque con esto se usan los métodos de proyectar, seleccionar o multiplicar.

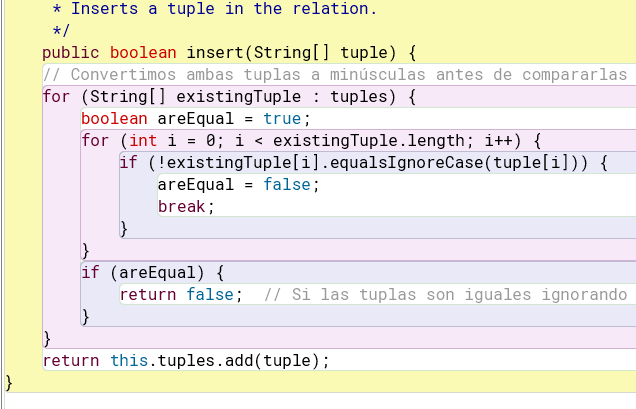
1. Implementen los métodos de Relation necesarios para pasar todas las pruebas definidas.

¿Cuáles métodos implementaron?

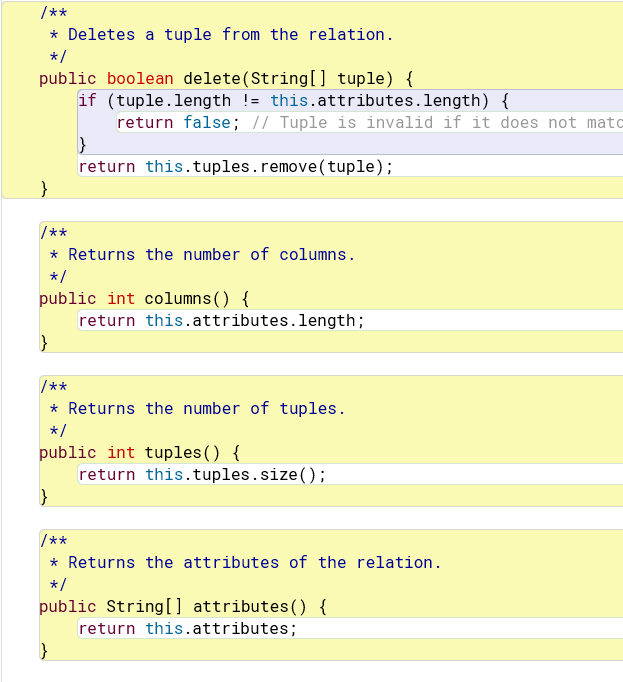
CONSTRUCTOR

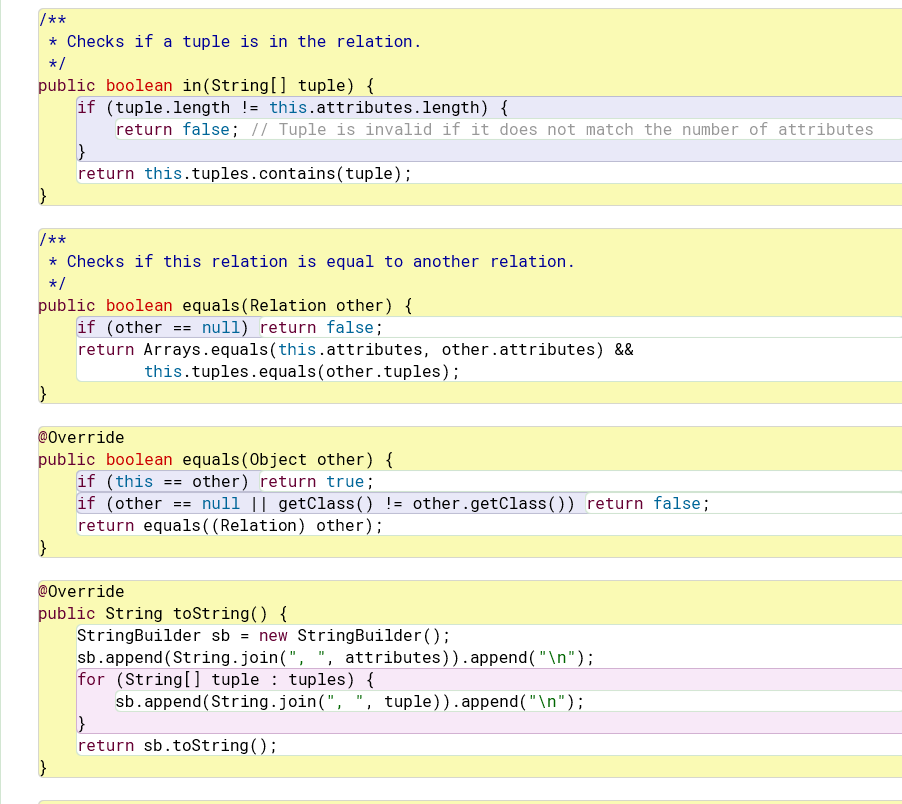


INSERT

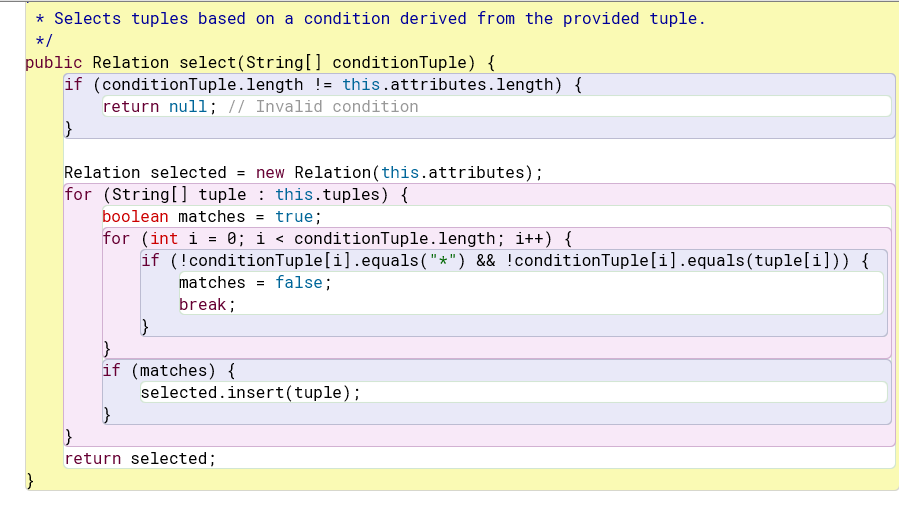


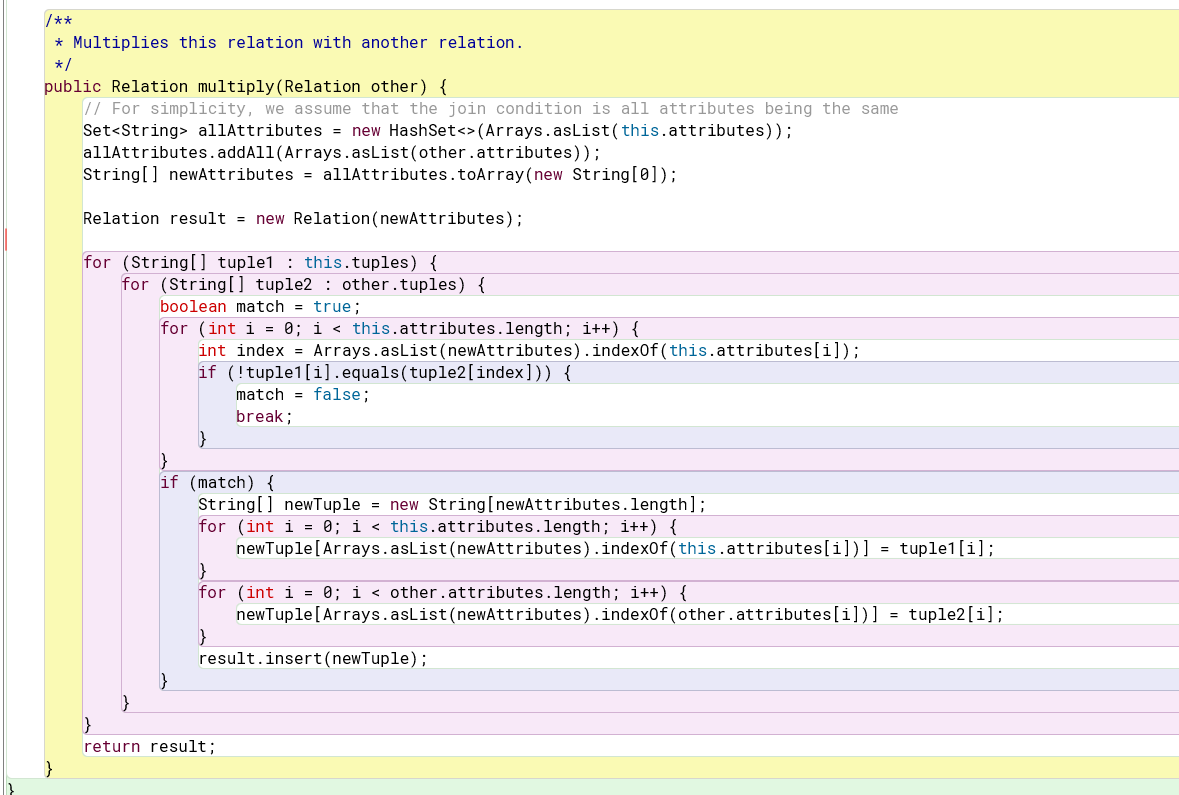
DELETE Y LOS OTROS MÉTODOS





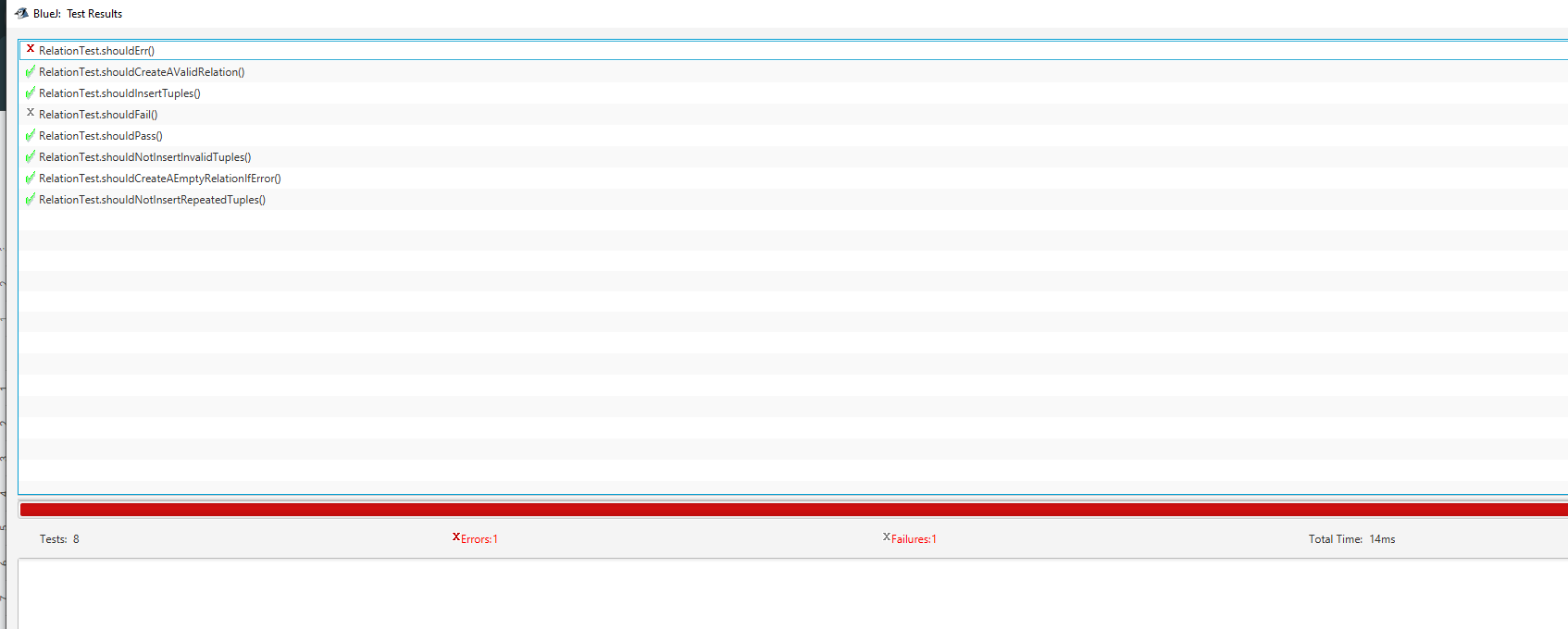






1. Capturen los resultados de las pruebas de unidad.





**Desarrollando relationalCalculator**

## BDD - MDD

[En lab02.doc, relationalCalculator.asta, \*.java]

Para desarrollar esta aplicación vamos a considerar algunos mini-ciclos. En cada mini-ciclo deben realizar los pasos definidos a continuación.

**MINI-CICLOS ACTUALES**

Ciclo 1 : Operaciones básicas de la calculadora: crear una calculadora y asignar, consultar e imprimir variables.

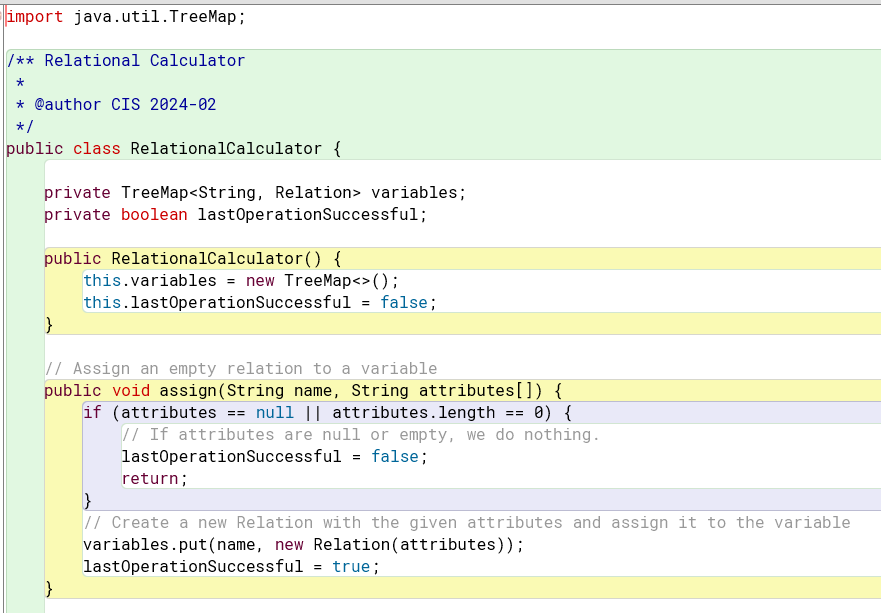
Ciclo 2 : Operaciones de manipulación de datos: insertar y eliminar tuplas en una relación.

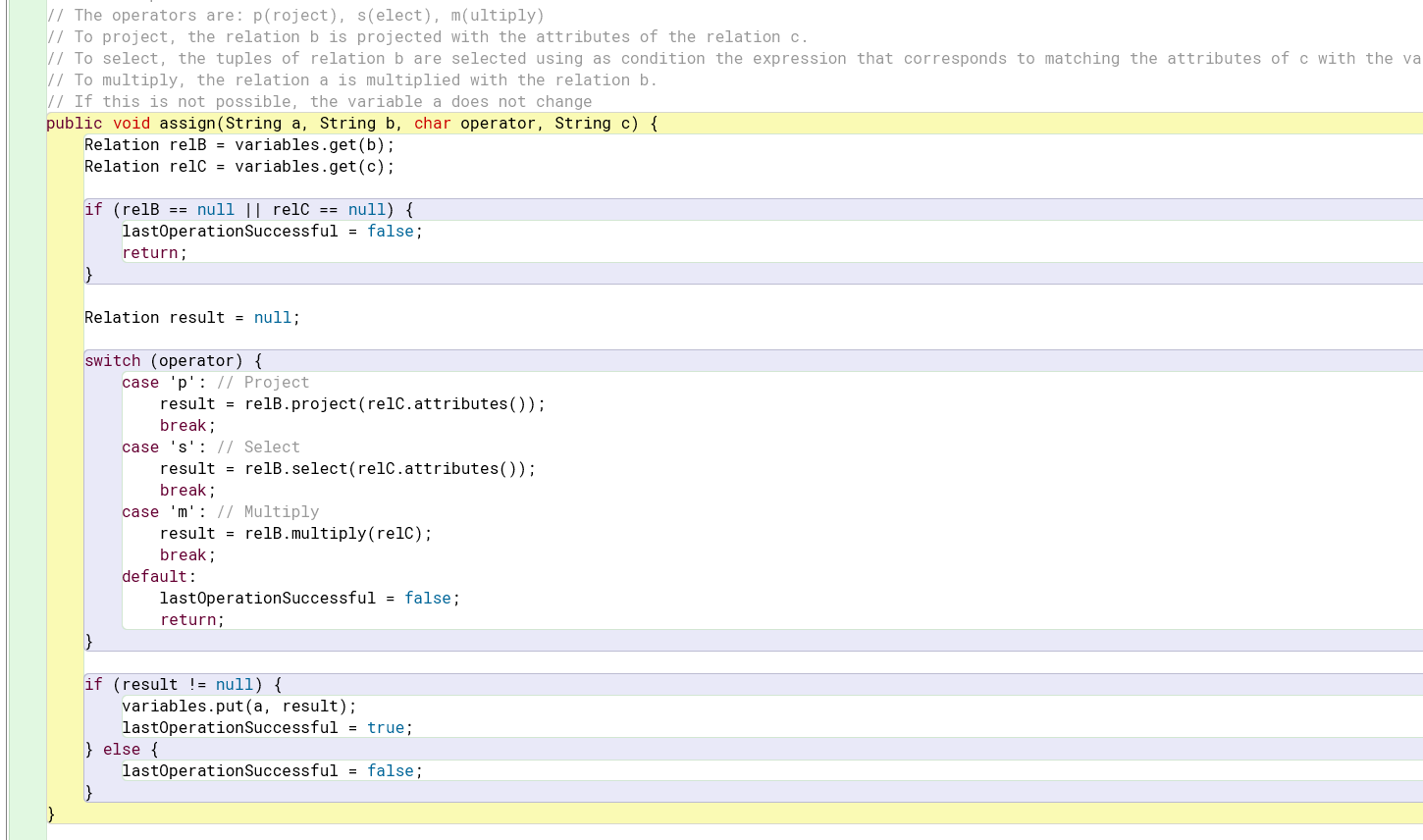
Ciclo 3 : Operaciones de consulta de datos: seleccionar, proyectar y multiplicar relaciones.

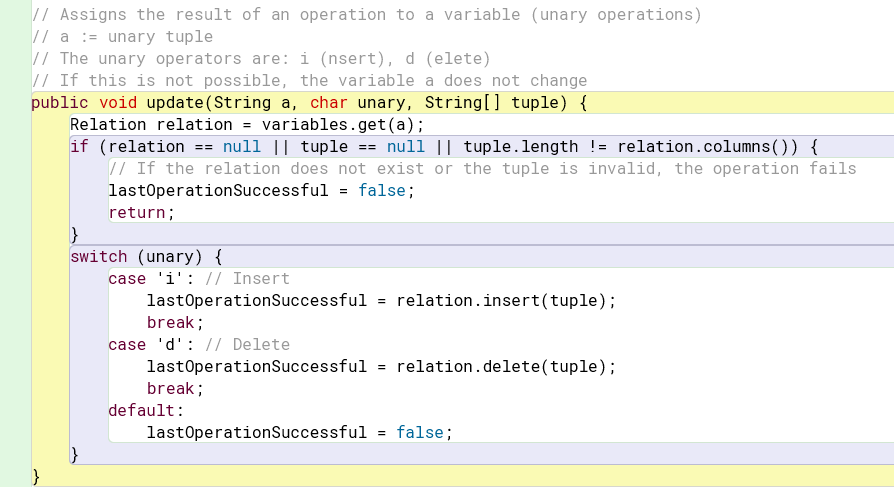
**BONO** Ciclo 4 : Defina dos nuevas funcionalidades. \*\*

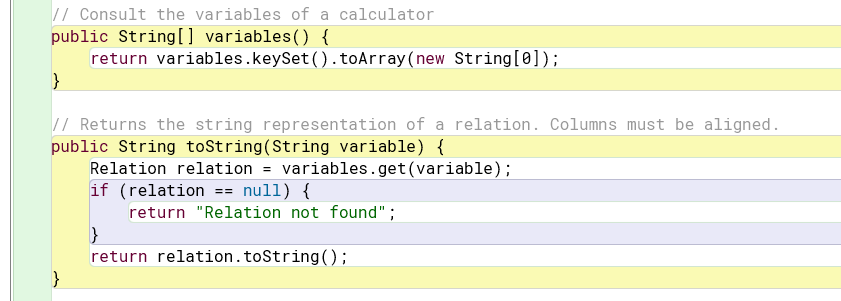
## Definir los métodos base correspondientes al mini-ciclo actual.

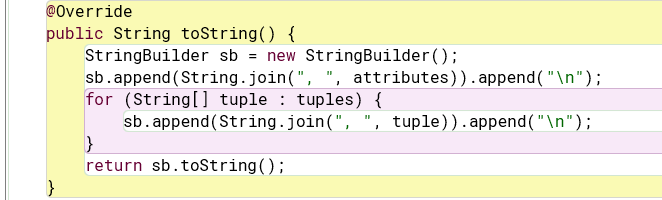
En relationalCalculator se codifican los métodos que tenían comentarios sobre lo que debería hacer, y que sea correctamente estructurado con documentación.







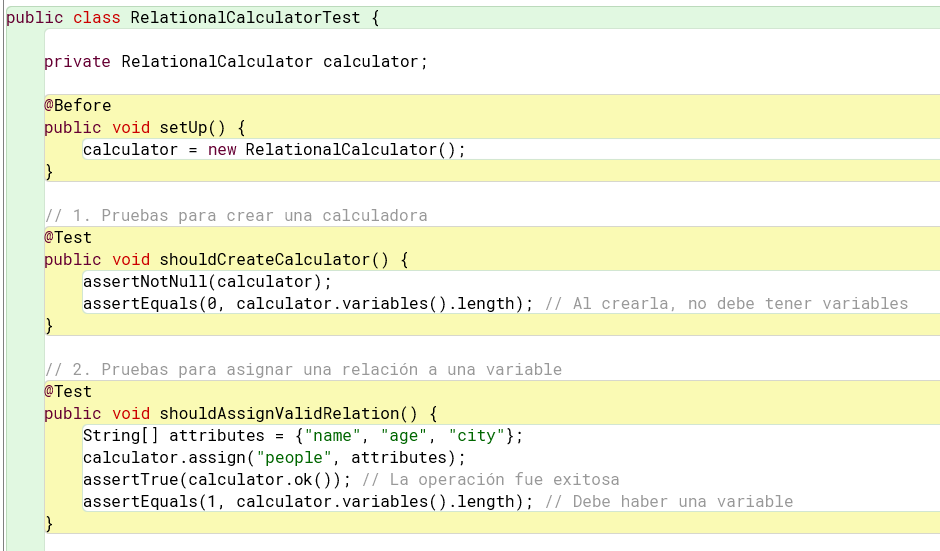


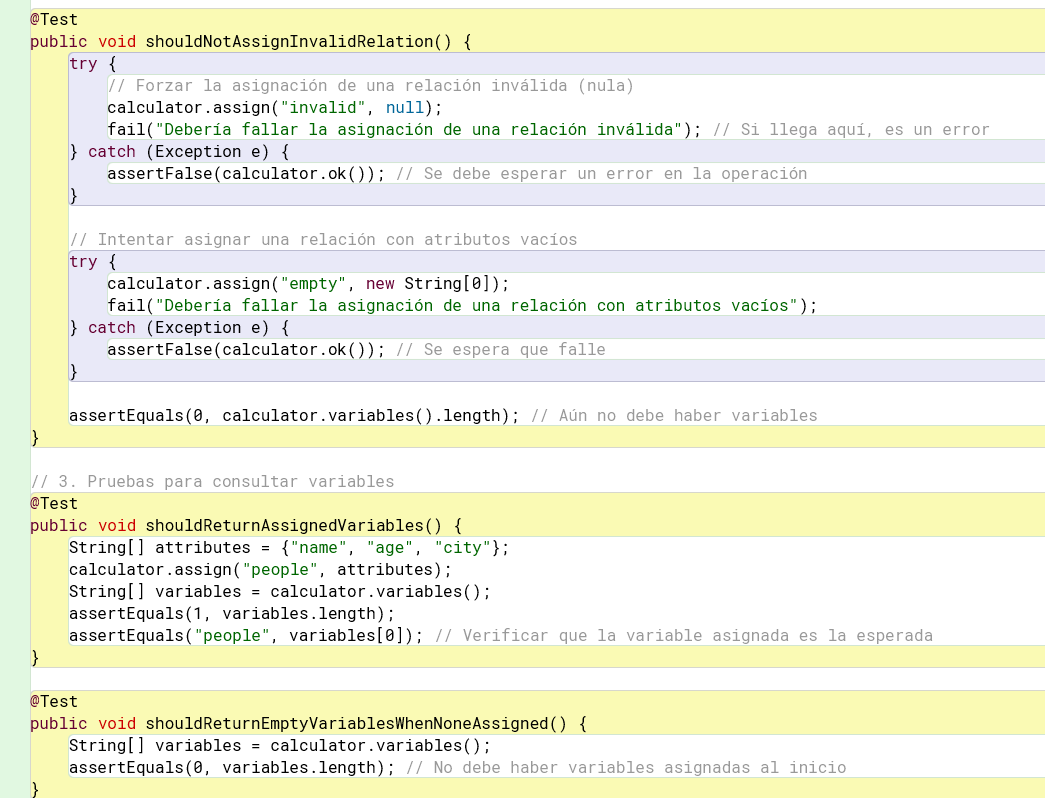


1. **Definir y programar los casos de prueba de esos métodos**

**Piensen en los deberia y los noDeberia (should and shouldNot)**

**EN ESTE PUNTO SE MUESTRA EL CÓDIGO TANTO DE SHOULD COMO DE SHOULD NOT, APLICADO AL MINI-CICLO 1, MÉTODOS BASE**

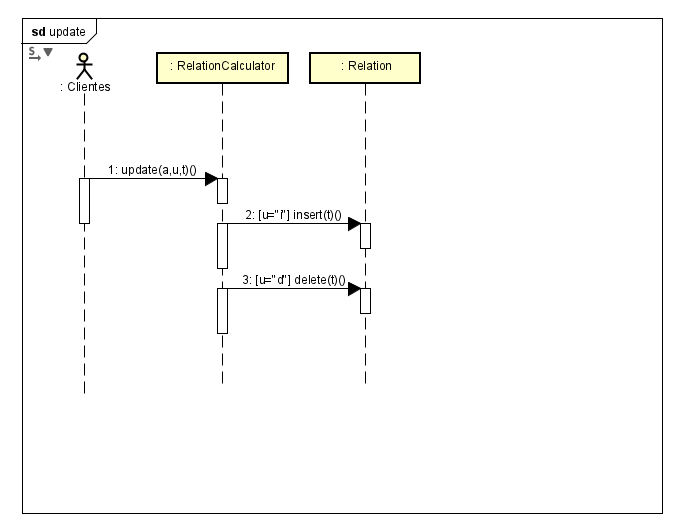
****

****

****

## Diseñar los métodos

**Usen diagramas de secuencia. En astah, adicione el diagrama al método.**

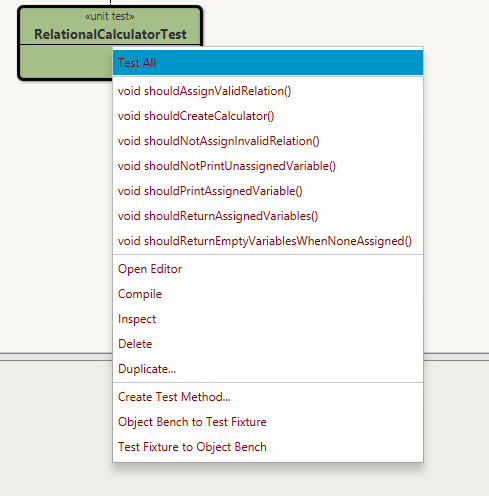
****

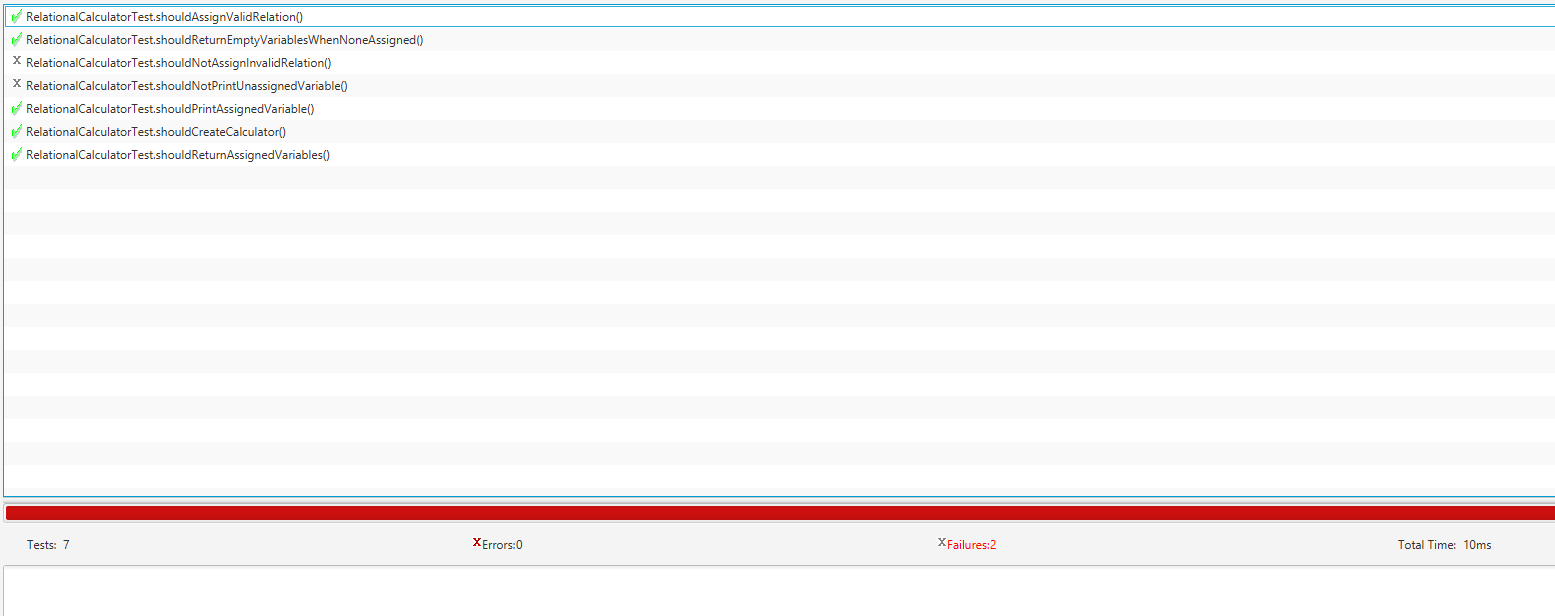
1. **Escribir el código correspondiente (no olvide la documentación)**

**EN EL ARCHIVO ZIP SE DETALLA QUE CADA MÉTODO ESTÁ DOCUMENTADO Y TAMBIÉN COMENTADO CON BASE A LAS CLASES CREADAS, SE PUEDE EVIDENCIAR AL GENERAR LA DOCUMENTACIÓN GENERALMENTE:**

****

1. **Ejecutar las pruebas de unidad (vuelva a 3 (a veces a 2), si no están en verde)**

****

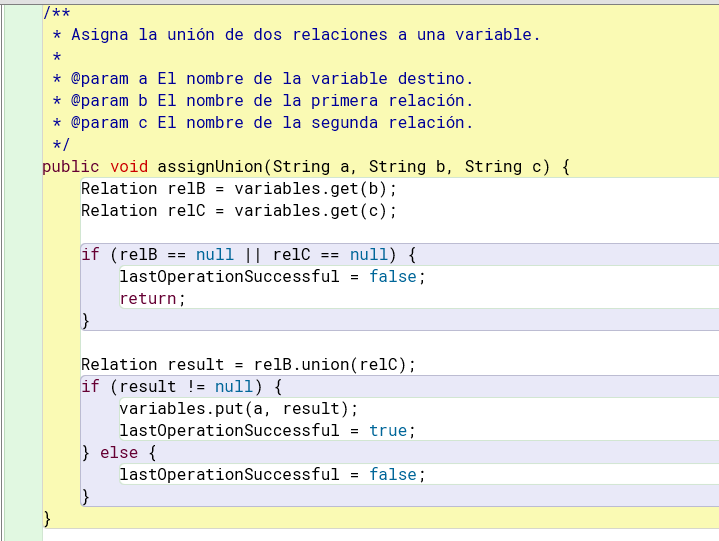
****

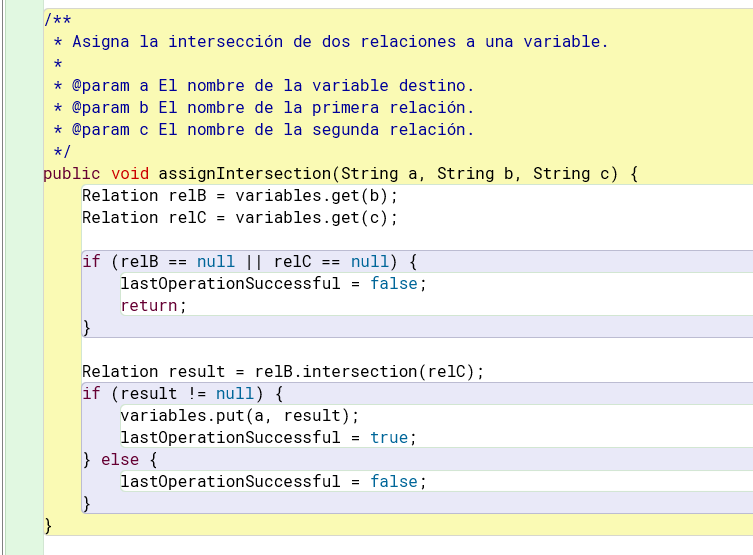
1. **Completar la tabla de clases y métodos. (Al final del documento)**

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

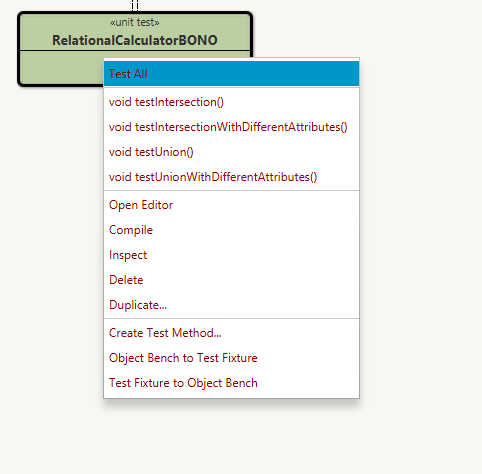
|  |  |  |
| --- | --- | --- |
| Mini-ciclo | relationalCalculator | relationalCalculatorTest |
| 1  2  3  4 BONO | RelationalCalculator(), assign(String, String[])  update(String, char, String[]),assign(String, String, char, String)  variables(), toString(String), ok()  assignUnion(), assignIntersection() | setUp(), shouldCreateCalculator(), shouldAssignValidRelation,  shouldNotAssignInvalidRelation()  shouldReturnEmptyVariablesWhenNoneAssigned(),  shouldReturnAssignedVariables(),shouldPrintAssignedVariable(),shouldNotPrintUnassignedVariable()  testUnion(), testIntersection(),  testUnionWithDifferentAttributes(), testIntersectionWithDifferentAttributes() |

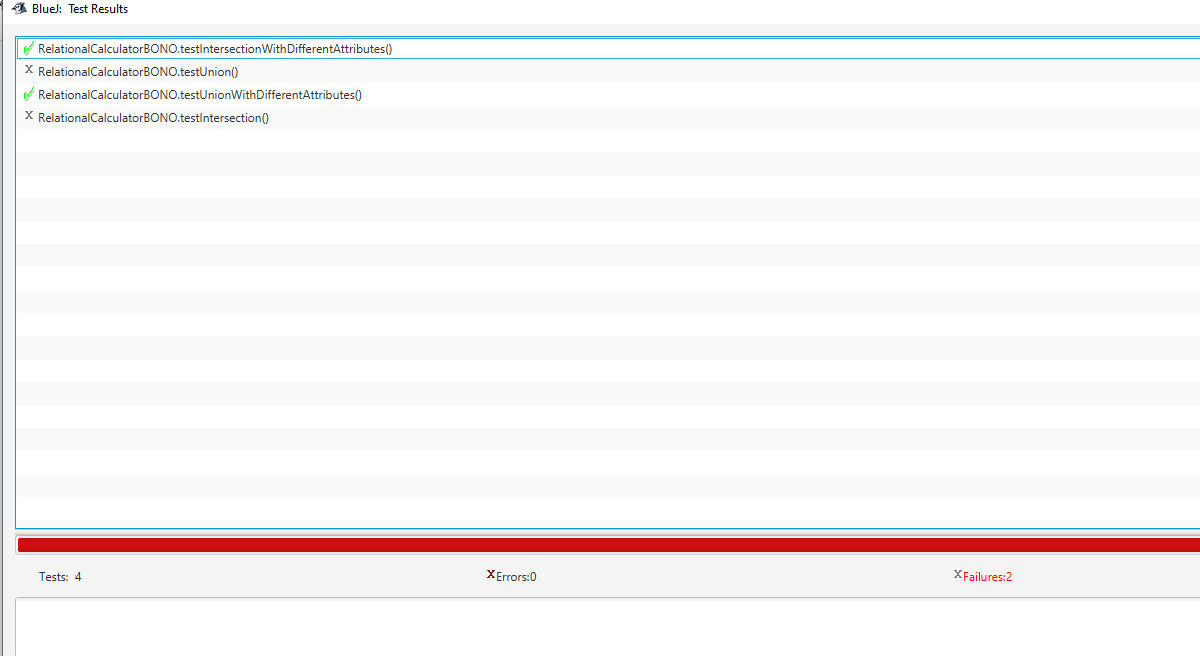
# El bono consiste en los métodos Union e Intersection, aplicado a diferentes pruebas donde se evidencia que sirven:





TESTS





# RETROSPECTIVA

1. ¿Cuál fue el tiempo total invertido en el laboratorio por cada uno de ustedes? (Horas/Hombre)

El tiempo total invertido fue de aproximadamente 20 horas, distribuidas a lo largo de varios días. Trabajamos principalmente en Pair Programming, lo que permitió que siempre uno pudiera aportar nuevas ideas mientras el otro programaba.

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

El laboratorio está completo, ya que cumplimos con todos los requisitos establecidos y realizamos todas las pruebas necesarias. Esto fue posible gracias a la buena organización en equipo y a la persistencia para resolver los problemas técnicos que surgieron. También utilizamos metodologías ágiles, lo cual nos ayudó a mantener el ritmo del trabajo.

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

La práctica más útil fue Pair Programming, ya que permitió que ambos miembros del equipo intercambiaran ideas de manera constante y que uno detectara errores o áreas de mejora en tiempo real mientras el otro programaba. Este enfoque aumentó nuestra eficiencia y evitó que pequeños errores pasaran desapercibidos.

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

El mayor logro fue terminar el laboratorio a tiempo y con todas las pruebas pasando satisfactoriamente. Esto fue especialmente gratificante, ya que en algunos puntos del desarrollo enfrentamos problemas que nos hicieron pensar que no llegaríamos a cumplir con los objetivos a tiempo. Sin embargo, con trabajo constante y comunicación efectiva, logramos superar esos obstáculos.

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

El mayor problema técnico fue implementar correctamente las operaciones de project y select en la clase Relation. Tuvimos dificultades para asegurar que los datos se ajustaran a los atributos seleccionados y que las tuplas cumplieran con las condiciones de selección. Para resolverlo, realizamos pruebas unitarias exhaustivas, ajustamos la lógica de indexación y comparación de datos, y refinamos el código hasta obtener los resultados correctos y eficientes.

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

Como equipo, hicimos bien en mantenernos organizados y enfocados. La comunicación fue clave, y el uso de herramientas como **Notion** y **Git** facilitó la colaboración en tiempo real. Para mejorar, nos comprometemos a dedicar más tiempo a la planificación previa y a realizar una investigación más profunda antes de implementar soluciones, lo que nos permitirá evitar algunos errores técnicos que enfrentamos al inicio.

**REFERENCIAS**

* <http://junit.org/javadoc/latest/>