PROGRAMACIÓN ORIENTADA A OBJETOS

## Diseño y Pruebas. Interacción entre objetos.

## 2022-1

## Laboratorio 2

## Andrés Camilo Oñate Quimbayo, Nicolas Ariza Barbosa

## A. Conociendo el proyecto

1. El proyecto BlueJ “SetCalculator” 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 contiene archivos con extensiones .class, .java, .ctxt, .bueJ, .TXT en el directorio principal, en la carpeta “doc” hay archivos con extensiones .html y .txt, en una carpeta contenida por esta última “resources” hay un archivo con extensión .gif

1. Exploren el proyecto en BlueJ

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

En total se tienen 3 clases *SetTest, Set y SetCalculator.* La relación entre estas clases es que tanto como *SetTest* como *SetCalculator* poseen una relación de atributos con la clase *Set,* funcionando esta última como base de para las demás clases en cuanto a las operaciones de conjuntos.

* ¿Cuál es la clase principal para la aplicación? ¿Cómo la reconocen?

La clase principal es SetCalculator, dado que es la encargada de realizar las operaciones a fin al propósito de la aplicación.

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

Existe una única clase diferente en este proyecto. Dicha clase es *SetTest* y está encargada de probar las características de la clase *SetCalculator* con un total de 6 testeos.

Para los siguientes 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)
   * Clase Set:

No tiene documentación en el encabezado ni en los métodos.

* + Clase SetCalculator:

Encabezado documentado, métodos sin documentar.

1. Revisen las fuentes del proyecto, ¿en qué estado está cada clase? (Detallen el estado de las fuentes considerando: código, documentación y comentarios) ¿Qué son el código, la documentación y los comentarios?

El **código** son las instrucciones ordenadas de un programa, lo que determina como se usa y manipula la información en búsqueda de realizar una tarea en específico.

La **Documentación** es una descripción detallada del código de nuestro programa. Que nos habla sobre las diferentes implementaciones de un programa, parámetros y funciones dentro de la implementación.

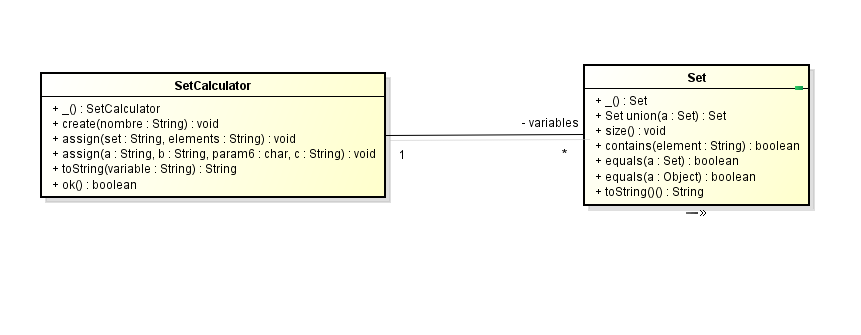
Los **comentarios** son líneas en el código que nos ayudan a describir en específico que hace una instrucción en concreto. Muy útiles a la hora de cubrir lo que la documentación no puede detallar.

* *SetTest* tiene su código completo, no posee comentarios ni tampoco posee documentación apropiada.
* *Set* en su código carece de atributos, no posee comentarios ni tampoco posee documentación apropiada.
* *SetCalculator* en el código está incompleta ya que casi todos sus métodos no realizan operaciones si no que retornan valores por defecto, no posee comentarios ni tampoco posee documentación apropiada.

## B. Ingeniería reversa

### MDD MODEL DRIVEN DEVELOPMENT

1. Realicen el diagrama de clases correspondiente a SetCalculator



1. . ¿Cuál contenedor está definido? Consulte la especificación y el API Java ¿Qué diferencias hay entre el nuevo contenedor y el ArrayList que conocemos?

HashMap:

“Hash table based implementation of the Map interface. This implementation provides all of the optional map operations, and permits null values and the null key. (The HashMap class is roughly equivalent to Hashtable, except that it is unsynchronized and permits nulls.) This class makes no guarantees as to the order of the map; in particular, it does not guarantee that the order will remain constant over time.”

ArrayList:

Resizable-array implementation of the List interface. Implements all optional list operations, and permits all elements, including null. In addition to implementing the List interface, this class provides methods to manipulate the size of the array that is used internally to store the list. (This class is roughly equivalent to Vector, except that it is unsynchronized.)

* Para acceder a los elementos en ArrayList se hace uso de un índice, mientras que en HashMap el índice puede ser de cualquier tipo de valor.
* El ArrayList es una colección ordenada (también conocida como secuencia). El usuario de esta interfaz tiene un control preciso sobre en qué lugar de la lista se inserta cada elemento. En un mapa se tiene pares clave/valor

Fuente: https://www.iteramos.com/pregunta/55974/la-diferencia-entre- lista-hashmap-y-matriz-en-java

## C. Conociendo Pruebas en BlueJ

### TDD -> BDD (TEST -> BEHAVIOUR DRIVEN DEVELOPMENT)

1. Revisen el código de la clase *SetTest.* ¿Cuáles etiquetas tiene (componentes con símbolos @)? ¿Cuántos métodos tiene? ¿Cuántos métodos son de prueba? ¿Cómo los reconocen?

Hay tres etiquetas *@Before, @Test y @After.* En total posee 8 métodos, 6 de ellos son de prueba y se reconocen por su despectivo nombre de *Test* haciendo idea de que son los métodos encargados de enviar las diferentes pruebas hacia otra clase.

1. Ejecuten los test de la clase SetTest. (click derecho sobre la clase, Test All) ¿cuántas pruebas se ejecutan? ¿cuántos pasan las pruebas? ¿por qué?

Se ejecutaron 6 pruebas, solo uno paso la prueba: “shouldCreateEmptySets()”.

Haciendo uso del metodo “assertEquals” de la clase Assert, la prueba comprueba que el método “size” de la clase Set devuelva el valor de 0.

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

*@Before:* Describe a un llamado previo de una clase en concreto la cual siempre se ejecuta antes de realizar una prueba o el llamado de cada uno de los métodos de prueba con la etiqueta *@Test.*

*@Test:* Describe a los métodos asociados a las pruebas que se le realizaran a una clase en concreto. Enviando diferentes valores y evaluando casos de prueba para verificar que nuestro código y programa funcionan.

*@After:* Describe al llamado que se hace tras finalizar una prueba en singular realizada por alguno de los métodos asociados con la etiqueta *@Test.*

1. Estudie los métodos assertTrue, assertFalse, assertEquals, assertArrayEquals, assertNull y fail de la clase Assert del API JUnit 2. Explique en sus palabras que hace cada uno de ellos.

* assertTrue: Comprueba que una condición pasada por parámetro es cierta.
* assertFalse: Comprueba que una condición pasada por parámetro es falsa.
* assertEquals: Comprueba que dos objetos son iguales.
* AssertArrayEquals: Comprueba que dos objetos array son iguales.
* assertNull: Comprueba que un objeto es NULL.
* fail: Fuerza que el test termine en fallo. Se puede mostrar un mensaje.

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

Los *fallos* se producen en Junit cuando alguno de los casos de prueba con los que estamos probando alguna clase/objeto arrojan un resultado contrario al esperado por el método de prueba.

Los *errores,* por otro lado, son discrepancias que se presentan al ejecutar nuestras pruebas, es decir, errores en la estructura del código (errores de compilación, errores de tipo apuntadores, mala sintaxis en el código etc.)

*Referencia*

*https://www.generacodice.com/es/articolo/631783/cu-l-es-la-diferencia-entre-el-fracaso-y-el-error-en-junit*

*Texto

Descripción generada automáticamenteInterfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente*

## D. Practicando Pruebas en BlueJ

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

* Definimos el atributo que es de tipo , con este atributo podemos obtener acceso directo a los elementos que componen al conjunto y además obtener información de este.

2. Indiquen los invariantes de la clase Set. Justifique la decisión.

* Los conjuntos siempre están ordenados.
* Las letras siempre estarán en mayúscula
* No se pueden agregar elementos individuales a un conjunto directamente.

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

* Se completo el constructor de la clase.
* Métodos implementados:

Interfaz de usuario gráfica

Descripción generada automáticamente

## E. Desarrollando SetCalculator

### BDD – MDD

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

Ciclo 1 : Operaciones básicas de la calculadora: declarar, asignar un valor y consultar Ciclo 2 : Operaciones binarias básicas: asignar el resultado de union e intersección Ciclo 3 : Operaciones binarias de resta: diferencia y diferencia simétrica

Ciclo 4 : Operaciones binarias avanzadas: producto cartesiano

Ciclo 5 : Defina una nueva funcionalidad.

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

## Ciclo 1

1. Definir los métodos base de correspondientes al ciclo actual.

-

-

-

-

1. Generar y programar los casos de prueba

* shouldCreateVariables()
* shouldToString()
* shouldToString()
* shouldAssing()
* shouldToOk()

1. Diseñar los métodos

Diagrama

Descripción generada automáticamente

Diagrama

Descripción generada automáticamente

Diagrama, Tabla

Descripción generada automáticamente

Diagrama

Descripción generada automáticamente

Diagrama

Descripción generada automáticamente

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

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

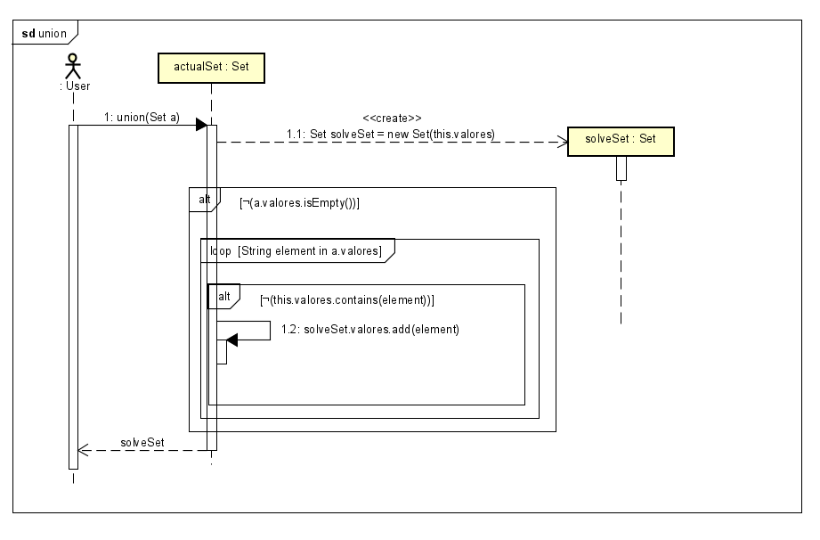
Descripción generada automáticamente

## Ciclo 2

1. Definir los métodos base de correspondientes al ciclo actual.
2. Generar y programar los casos de prueba

* shouldAssingOperationsUnion()
* shouldAssingOperationsIntersection()

1. Diseñar los métodos



Diagrama

Descripción generada automáticamente

## Ciclo 3

1. Definir los métodos base de correspondientes al ciclo actual.

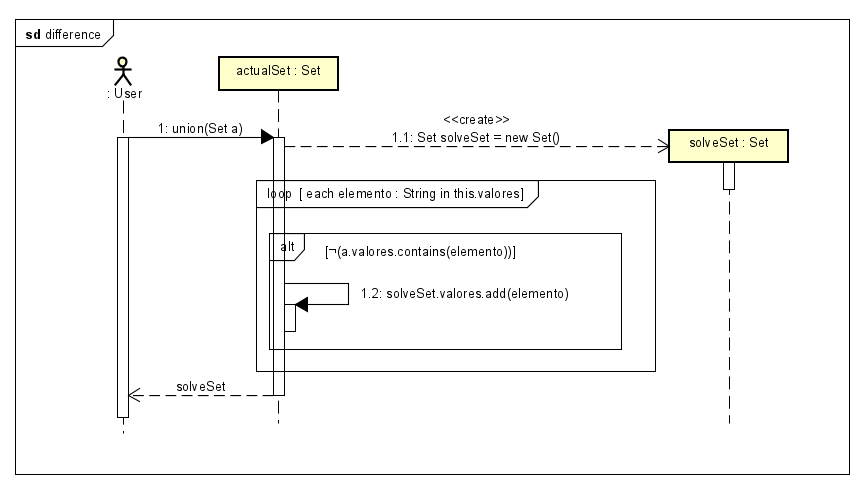
-

-

1. Generar y programar los casos de prueba

* shouldAssingOperationsDifference
* shouldAssingOperationsSymmetricDifference

1. Diseñar los métodos



Diagrama

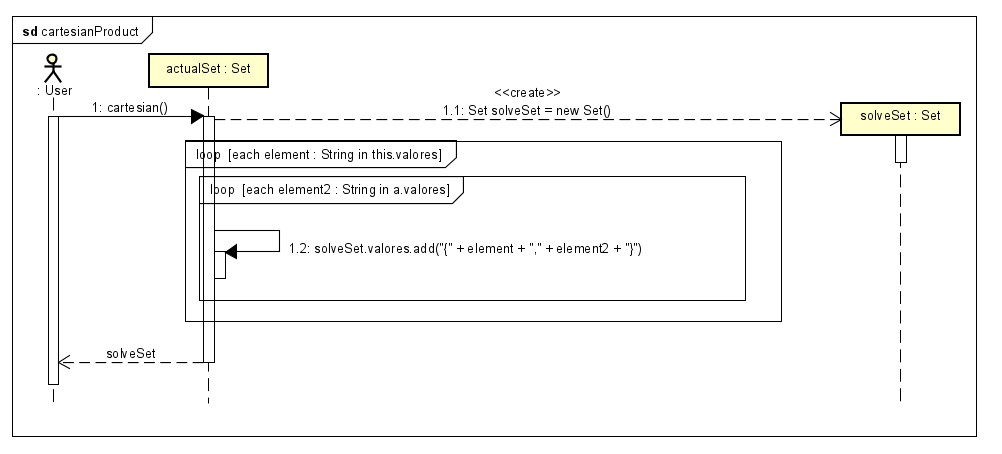
Descripción generada automáticamente

## Ciclo 4

1. Definir los métodos base de correspondientes al ciclo actual.
2. Generar y programar los casos de prueba

* shouldAssingOperationsProduct()

1. Diseñar los métodos

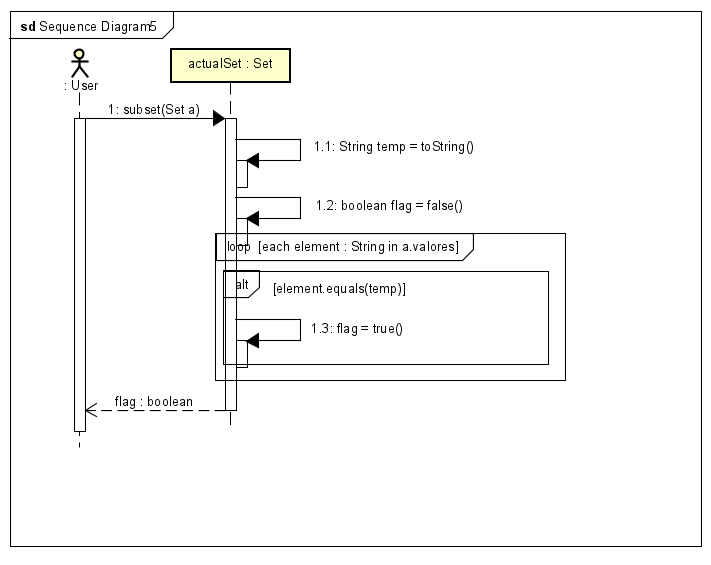


## Ciclo 5

1. Definir los métodos base de correspondientes al ciclo actual.
2. Generar y programar los casos de prueba

* shouldAssingOperationsSubset ()

1. Diseñar los métodos



|  |  |  |
| --- | --- | --- |
| Ciclo | SetCalculator | SetCalculatorTest |
| 1 | create | shouldCreateVariables (Debería crear variables) |
|  | assign | shouldAssing(Debería asignar a una variable existente) |
|  | toString | shouldToString(Debería dar el valor asociado a una variable) |
|  | ok | shouldToOk() (Debería responder a una operación ) |
| 2 | union | shouldAssingOperationsUnion() (Debería unir Sets) |
|  | intersection | shouldAssingOperationsIntersection() (Debería inter.. Sets) |
| 3 | diference | shouldAssingOperationsDifference (Debería - Sets) |
|  | symmetricDifference | shouldAssingOperationsSymmetricDifference () (Debería \_ Sets) |
| 4 | cartesianProduct | shouldAssingOperationsProduct (Debería x Sets) |
| 5 | subset | shouldAssingOperationsSubset () |

## RETROSPECTIVA

1. ¿Cuál fue el tiempo total invertido en el laboratorio por cada uno de ustedes? 12 (Horas/Hombre)
2. ¿Cuál es el estado actual del laboratorio? ¿Por qué?

Completo

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

CODE THE UNIT TEST FIRST y ALL CODE MUST HAVE UNIT TESTS

Agiliza el desarrollo del código.

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

Las operaciones entre conjuntos (unir, intersección ..)

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

Se cambio el método equals de la clase Set y generaba errores inesperados.

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

Buena coordinación en los tiempos de elaboración del laboratorio y mayor eficiencia en el desarrollo práctico del código y documento.