ESCUELA COLOMBIANA DE INGENIERÍA
PROGRAMACIÓN ORIENTADA A OBJETOS
Diseño y Pruebas.
Interacción entre objetos.
2021-1
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 :

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 el foro de entrega 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:
Las calculadoras de pila sirven para evaluar expresiones en notación postfija. Estas calculadoras adicionan los operandos a la pila y cuando llega un operador toma los últimos dos operandos de la pila, aplica el operador y coloca el resultado a la pila. Por ejemplo, para evaluar 1 2 3 + * pasaría por los siguientes estados: a) 1; b) 1 , 2 ; c) 1 , 2 , 3; d) 1 , 5; e) 5

En este laboratorio vamos a construir una calculadora de pila de cálculo relacional para tablas : relational Calculator

Conociendo el proyecto

1. El proyecto BlueJ “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.

Hay tres clases
RelationalCalculator.java Table.java ,TableTest.java las cuales al ser compiladas generan un archivo llamado igual que la clase, pero con la diferencia que su extensión será .class
ejemplo:
table.class, tableTest.class, RelationalCalculator.class

```
 Directorio de C:\Users\user\Desktop\Universidad\POOB 2021 ahora si\Lab 02\calculator

12/02/2021  11:32 a. m.    <DIR>          .
12/02/2021  11:32 a. m.    <DIR>          ..
11/02/2021  08:39 p. m.    <DIR>          doc
12/02/2021  06:50 a. m.             1.250 package.bluej
04/02/2012  09:49 p. m.               471 README.TXT
12/02/2021  06:50 a. m.             1.091 RelationalCalculator.class
12/02/2021  06:50 a. m.             1.220 RelationalCalculator.ctxt
12/02/2021  06:31 a. m.             1.145 RelationalCalculator.java
12/02/2021  06:50 a. m.             1.997 Table.class
12/02/2021  06:50 a. m.             1.810 Table.ctxt
12/02/2021  06:38 a. m.             2.086 Table.java
12/02/2021  06:50 a. m.             2.598 TableTest.class
12/02/2021  06:50 a. m.               832 TableTest.ctxt
12/02/2021  06:50 a. m.             2.099 TableTest.java
              11 archivos         16.599 bytes
               3 dirs  111.200.292.864 bytes libres

C:\Users\user\Desktop\Universidad\POOB 2021 ahora si\Lab 02\calculator>
```

2. Exploren el proyecto en BlueJ
¿Cuántas clases tiene?
Tres clases (table, tableTest, RelationalCalculator)
¿Cuál es la relación entre ellas?
La clase RelationalCalculator se relaciona con la clase Table y la clase TableTest es la que realiza las pruebas las cuales serán aplicadas por medio de la clase Table.
¿Cuál es la clase principal?
Table es la clase principal
¿Cómo la reconocen?
Se reconocen porque las pruebas se ejecutan sobre esta clase y también en el diagrama que nos da BlueJ se denota que table usa las otras dos clases.
¿Cuáles son las clases “diferentes”?
La clase diferente es: TableTest la cual usa la importación de “static org.junit.Assert.*” para realizar pruebas a los métodos de las clases que definamos.
¿Cuál es su propósito?
Una buena práctica en programación es probar que los métodos o ciclos que se están desarrollando funcionan correctamente y en cuestión de ahorro del tiempo se crean este tipo de clases para optimizar estar probando los métodos una y otra vez.

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 de cada clase: encabezado y métodos)

Encabezado:

Class Table

java.lang.Object
  Table

```
public class Table
extends java.lang.Object
```

Author:

ECI, 2021-1

Métodos:

Method Summary

All Methods | Instance Methods | Concrete Methods

Modifier and Type	Method and Description
int	cardinal() Inserts the specified tuples to this table
boolean	contenido(Table t)
Tabla	diferencia(Table t)
Tabla	diferenciaSimetrica(Table t)
boolean	equals(java.lang.Object o) Compara si esta relación es igual a otra (el parametro debe ser una relación)
boolean	in(java.lang.String[] tupla) Verifica si una tupla pertenece a la relación
Tabla	interseccion(Table t)
Tabla	producto(Table t)
java.lang.String	toString() Retorna una cadena que describe esta relación con los elementos entre parentesis, separados con cambio de línea y ordenados alfabéticamente componente a componente

La documentación para la clase RelationalCalculator aún no se encuentra disponible.

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

¿En qué estado está cada clase?

Ninguna clase está completa, ya que sus métodos están incompletos.

¿Qué son el código, la documentación y los comentarios?

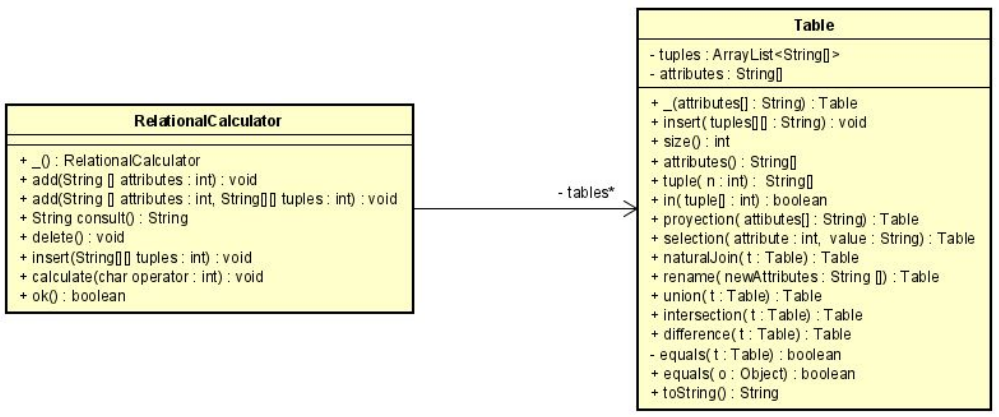
Código: Código se refiere al algoritmo de programación que está escrito en un lenguaje de programación.

Documentación: Se refiere a la acción de explicar un código, lo general del código, sus parámetros y su valor de retorno. La diferencia es que la documentación se genera a través de los comentarios. Es importante aclarar que solo se mostraran métodos, atributos que tengan el valor de Public, estos métodos, son los que por lo general usaran el usuario.

Comentarios: Un comentario sirve para explicar el funcionamiento de un fragmento de código.

Ingeniería reversa

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



2. ¿Qué tipo de contenedor está definido? Consulte la especificación y el API Java
¿Qué diferencias hay entre el Stack, ArrayList,[]?

String[] attributes;
ArrayList<String[]> tuples;
Stack<Table> tables;

STACK: La clase Stack representa una pila de objetos de último en entrar, primero en salir (LIFO). Extiende la clase Vector con cinco operaciones que permiten tratar un vector como una pila. Se proporcionan las operaciones habituales de push y pop, así como un método para mirar el elemento superior de la pila, un método para comprobar si la pila está vacía, y un método para buscar un elemento en la pila y descubrir a qué distancia está de la parte superior.

ARRAY: La clase Array proporciona métodos estáticos para crear y acceder dinámicamente a las matrices de Java.
Array permite que se produzcan conversiones de ampliación durante una operación de obtención o establecimiento, pero lanza una IllegalArgumentException si se produce una conversión de reducción.

ARRAYLIST: Implementación de una matriz redimensionable de la interfaz List. Implementa todas las operaciones opcionales de la lista y permite todos los elementos, incluidos los nulos. Además de implementar la interfaz List, esta clase proporciona métodos para manipular el tamaño de la matriz que se utiliza internamente para almacenar la lista. (Esta clase es aproximadamente equivalente a Vector, excepto que no está sincronizada).
Las operaciones size, isEmpty, get, set, iterator y listIterator se ejecutan en tiempo constante. La operación add se ejecuta en tiempo constante amortizado, es decir, añadir elementos requiere un tiempo(n). Todas las demás operaciones se ejecutan en tiempo lineal (a grandes rasgos). El factor constante es bajo comparado con el de la implementación de LinkedList.

Conociendo Pruebas en BlueJ

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

1. Revisen el código de la clase TableTest.
¿cuáles etiquetas tiene (componentes con símbolo @)?
@Before
@Test
@After
¿cuántos métodos tiene?
Hay 9 métodos.
setUp(),shouldPass(),shouldFail(), shouldErr(),shouldCreateAnEmptyTables(),shouldCreateTables(),shouldRepresentTableAsString()
,shouldNotInsertBadTuples(), tearDown().
¿cuantos métodos son de prueba?
Hay 7 métodos

shouldPass(),shouldFail(), shouldErr(), shouldCreateAnEmptyTables(), shouldCreateTables(), shouldRepresentTableAsString() , shouldNotInsertBadTuples()

¿cómo los reconocen?

por la etiqueta Test.

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

¿cuántas pruebas se ejecutan?

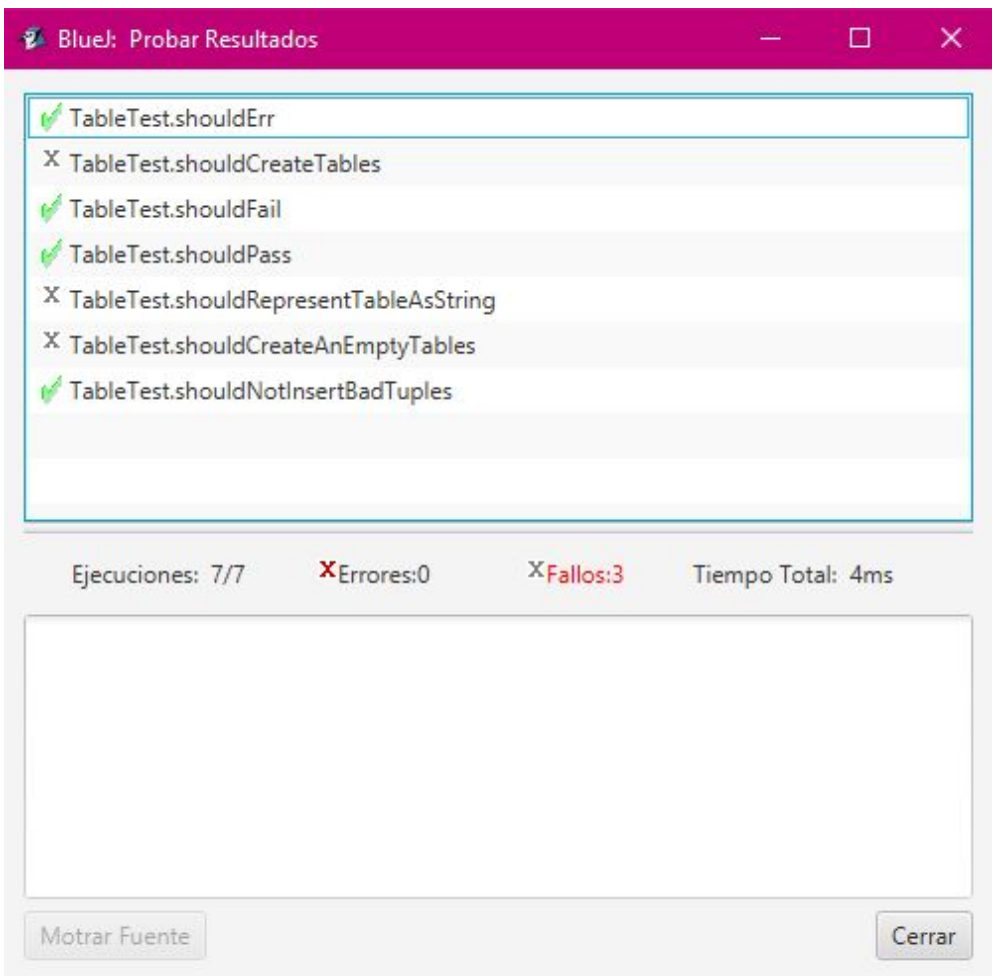
Se ejecutan todas las pruebas

¿cuántos pasan las pruebas?

solo pasan 4 pruebas

¿por qué?

Porque aún no están implementados los métodos de las clases.



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

@Before

Etiqueta que se usa para que su ejecución en la clase de prueba sea la primera. La cual nos permitirá o mejor dicho tener la ventaja de inicializar clases antes de probarla, así poder reutilizar el código de la clase test

@After

Etiqueta que sirve para decirle a una clase de pruebas que ese método se ejecutará de últimas.

@Test

La etiqueta Test ordena ejecutar los métodos instanciados de tipo .test para validar un resultado

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

assertTrue: Método que al recibir como parámetro el valor booleano de true, devuelve el test como correcto.

assertFalse:Método que al recibir como parámetro el valor booleano de false, devuelve el test como correcto.

assertEquals: Método recibe como parámetro dos cosas: valor x y valor y, donde si x=y entonces devuelve el test como correcto.

assertArrayEquals: Método recibe como parámetro dos cosas: un array x y un array y donde si x contiene lo mismo que y devuelve el test como correcto.

assertNull: Método que al recibir como parámetro un valor , devuelve el test como correcto si el valor es “null”.

Fail: Es un método que no pasa la prueba con el parámetro dado.

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

Si una condición afirmada no es verdadera, la prueba se detiene allí y JUnit informa una falla en la prueba. También es posible que cuando JUnit ejecute la prueba, se genere una excepción y no se capture, en este caso JUnit informa un error de prueba.

@Test

```
public void shouldPass(){
    assertTrue(true);
}

@Test
public void shouldFail(){
    assertTrue(false);
}

@Test
public void shouldErr(){
    int[] x={1,2,3};
    int[] y={};
    for (int i=0; i<3;i++){
        assertEquals(x[i],y[i]);
    }
}
```

Practicando
Pruebas en BlueJ

Ahora vamos a escribir el código necesario para las pruebas de TableTest.

1. Determinen la forma en que van a almacenar los elementos de un conjunto. Justifique la selección. Para esto revisen el conjunto de pruebas definido hasta el momento.

Para almacenar los elementos de un conjunto necesitaremos una lista de strings la cual va a contener los elementos de las columnas de la clase Table y necesitaremos un arreglo de strings que va a contener los datos que contiene ese conjunto.

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

```
/**
 * Inserts the specified tuples to this table
 * @param tuples,
**/
public void insert(String tuples1[][]) {
    for (int i=0;i<tuples1.length;i++)
    {
        tuples.add(tuples1[i]);
    }
}

public int size(){
    return tuples.size();
}

public boolean in(String tuple[]){
    return tuples.contains(tuple);
}

public String[] attributes(){
    return attributes;
}
```

```java
@Override
public String toString () {
      String s = "[";
      for (int i=0; i<attributes.length;i++ )
      {
          s+=attributes[i].toUpperCase();
          if (i !=(attributes.length -1)){s+=",";}
      }
      s+=")";
      System.out.println(s);
      for (int i=0; i<tuples.size();i++ )
      {
        s +="\n(";
          for(int j=0;j<tuples.get(i).length;j++){
          s+=tuples.get(i)[j];
          if (j !=(tuples.get(i).length-1)){s+=",";}
        }
        s+=")";

      }
      s+="\n";
      System.out.println(s);
      return s;
}
```

Desarrollando RelationalCalculator BDD - MDD

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

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

Ciclo 1 : Operaciones básicas de pila: crear, adicionar y consultar

Ciclo 2 : Operaciones relacionales básicas: insertar, seleccionar, proyectar

Ciclo 3 : Operaciones relacionales avanzadas: juntar, renombrar

Ciclo 4 : Operaciones básicas de conjuntos : unir, intersectar, restar

Ciclo 5 : Defina una nueva funcionalidad.

```java
//metodo creado por el equipo
public void mostrar()
{
    for (int i=0;i<attributes.length;i++){System.out.print(attributes[i]+"    ");}
    System.out.println();
    for (int i=0;i<attributes.length;i++){for (int j=0;j<tuples.size();j++){System.out.print(tuples.get(i)[j]+"        ");}System.out.println();}
}
```

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

Ciclo	relationalCalculator	relationalCalculatorTest	Table	TableTest
1	add(String [] attributes)	DeberiaInsertar()		
1	add(String [] attributes, String[][] tuples)	DeberiaInsertar()		
1	size()			
1	consult()	DeberiaConsultar()	toString()	shouldRepresentTableAsString()

Ciclo	relationalCalculator	relationalCalculatorTest	Table	TableTest
2			insert()	DeberiaInsertar()
2			selection(String attribute,String value)	ShouldSelectionGood()
2			proyection(String attribues2[])	ShouldProyectionGood()
3			rename()	ShouldChangeName)
5			mostrar()	Shouldmostrar()

RETROSPECTIVA

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

20 horas cada uno

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

Casi completo, se dificultaron algunos puntos a causa de entendimiento de algunos puntos

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

hacer los test antes de programar, eso nos daba una idea más clara a lo que debían llegar los métodos

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

Cuando la primera prueba del laboratorio corrió satisfactoriamente.

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

teníamos algunas dudas de algunos métodos que no eran claros, usamos el foro para resolver la duda y ante la ausencia de la respuesta consultamos con la profe irma.

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

Compartir las ideas para solucionar los problemas, trabajar la alternancia de roles.