Universidad del Valle de Guatemala
Facultad de Ingeniería
Departamento de Ciencias de la Computación
CC2003–Algoritmos y Estructura de Datos

Bryan Martínez 23542
Adriana Palacios 23044

Hoja de trabajo 4

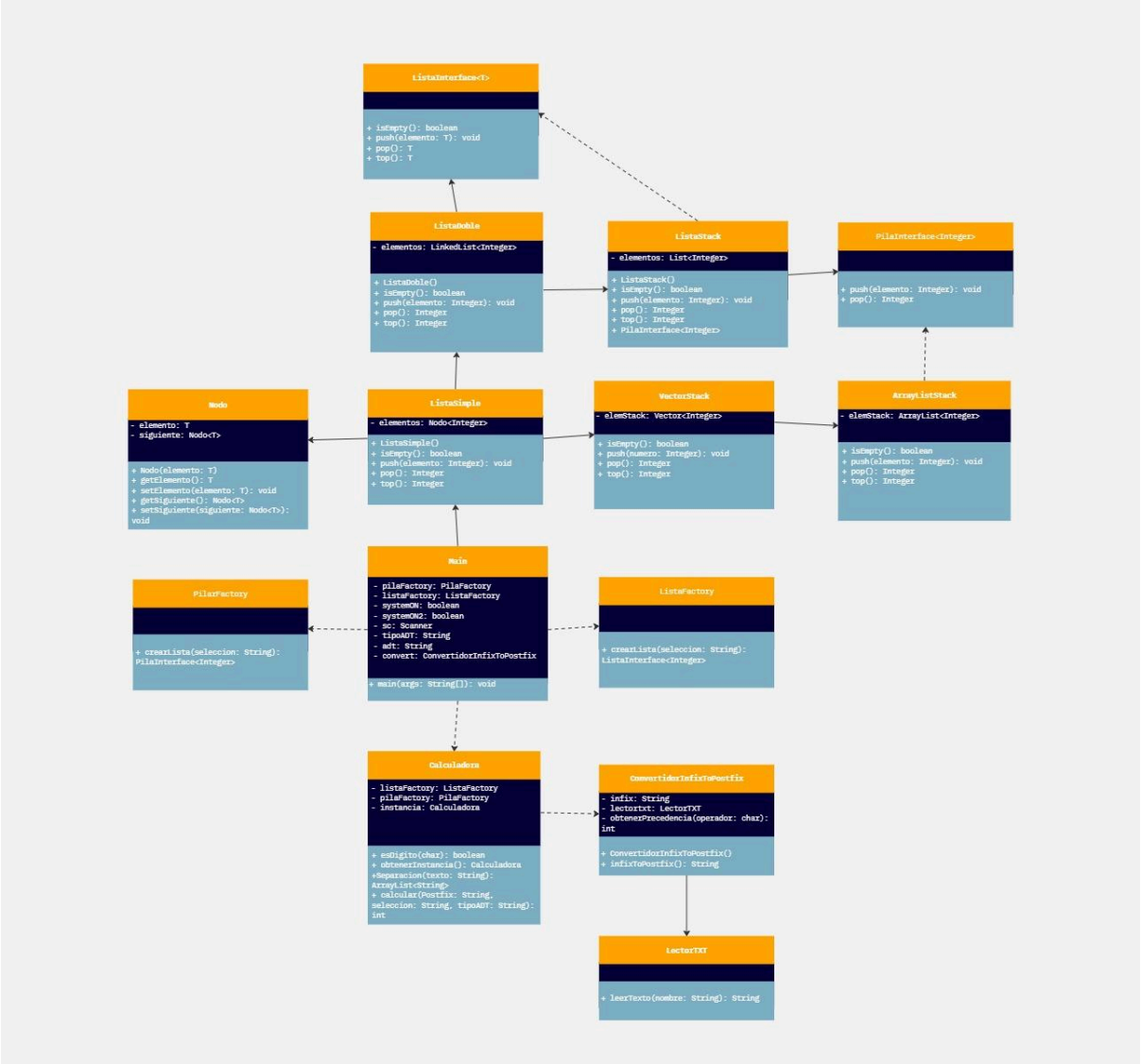
Link al repositorio con todas las evidencias:

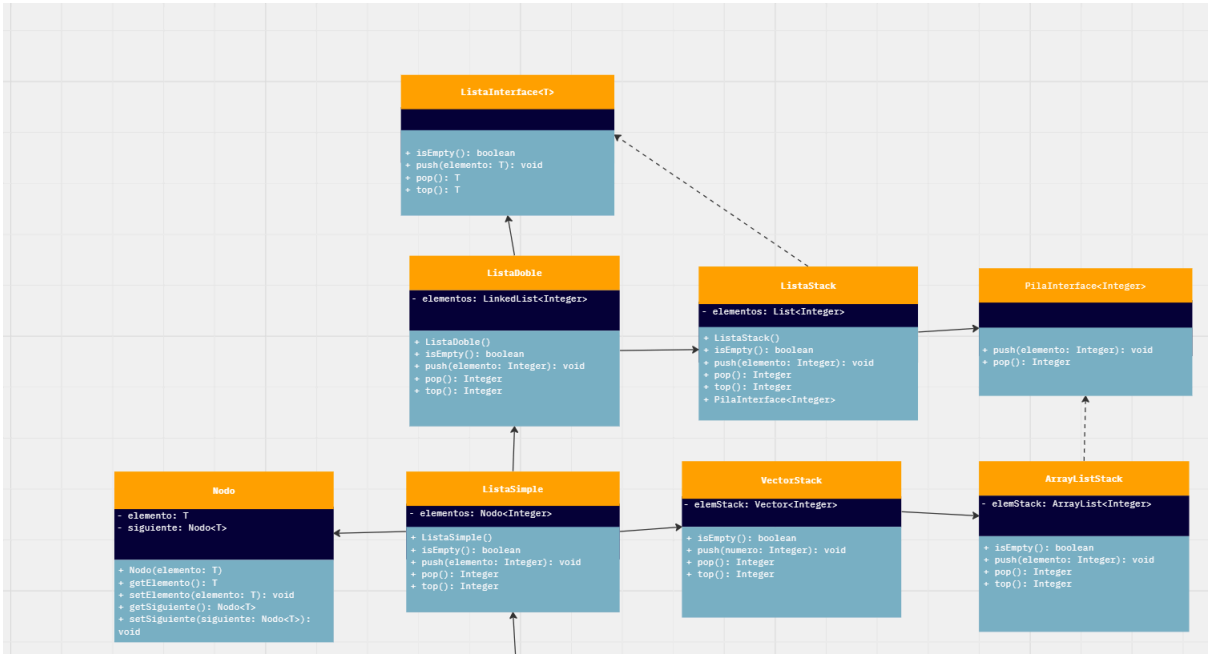
Link a GitHub

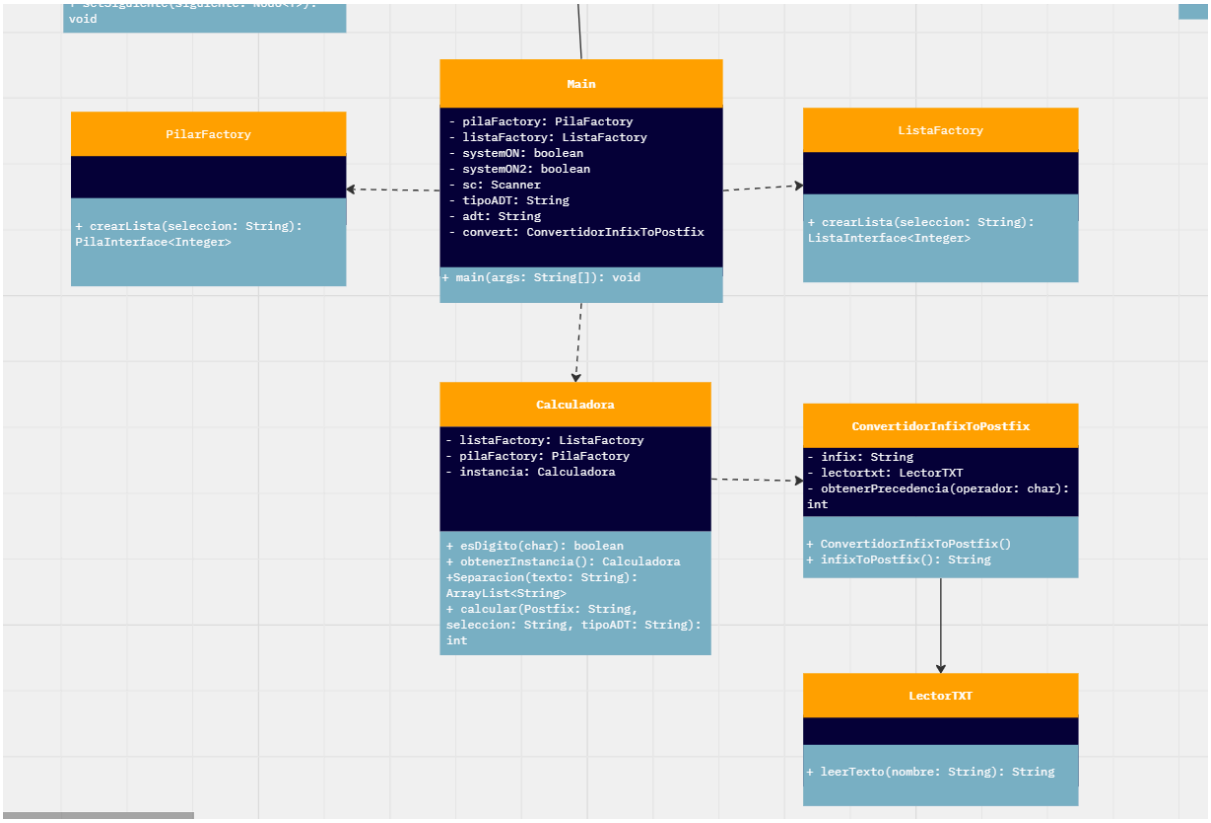
Link al repositorio con formato Maven (se tuvieron algunos inconvenientes, nada más se pegaron los archivos a Maven):

Link a GitHub (Versión Final de Entrega)

Diagrama UML:




ListaInterface<T>
+ isEmpty(): boolean
+ push(elemento: T): void
+ pop(): T
+ top(): T
ListaDoble
- elementos: LinkedList<Integer>
+ ListaDoble()
+ isEmpty(): boolean
+ push(elemento: Integer): void
+ pop(): Integer
+ top(): Integer
ListaStack
- elementos: List<Integer>
+ ListaStack()
+ isEmpty(): boolean
+ push(elemento: Integer): void
+ pop(): Integer
+ top(): Integer
+ PilaInterface<Integer>
PilaInterface<Integer>
+ push(elemento: Integer): void
+ pop(): Integer
Nodo
- elemento: T
- siguiente: Nodo<T>
+ Nodo(elemento: T)
+ getElemento(): T
+ setElemento(elemento: T): void
+ getSiguiente(): Nodo<T>
+ setSiguiente(siguiente: Nodo<T>): void
ListaSimple
- elementos: Nodo<Integer>
+ ListaSimple()
+ isEmpty(): boolean
+ push(elemento: Integer): void
+ pop(): Integer
+ top(): Integer
VectorStack
- elemStack: Vector<Integer>
+ isEmpty(): boolean
+ push(numero: Integer): void
+ pop(): Integer
+ top(): Integer
ArrayListStack
- elemStack: ArrayList<Integer>
+ isEmpty(): boolean
+ push(elemento: Integer): void
+ pop(): Integer
+ top(): Integer


void
Main
- pilaFactory: PilaFactory
- listaFactory: ListaFactory
- systemON: boolean
- systemON2: boolean
- sc: Scanner
- tipoADT: String
- adt: String
- convert: ConvertidorInfixToPostfix
+ main(args: String[]): void
PilarFactory
+ crearLista(seleccion: String): PilaInterface<Integer>
ListaFactory
+ crearLista(seleccion: String): ListaInterface<Integer>
Calculadora
- listaFactory: ListaFactory
- pilaFactory: PilaFactory
- instancia: Calculadora
+ esDigito(char): boolean
+ obtenerInstancia(): Calculadora
+Separacion(texto: String): ArrayList<String>
+ calcular(Postfix: String, seleccion: String, tipoADT: String): int
ConvertidorInfixToPostfix
- infix: String
- lectortxt: LectorTXT
- obtenerPrecedencia(operador: char): int
+ ConvertidorInfixToPostfix()
+ infixToPostfix(): String
LectorTXT
+ leerTexto(nombre: String): String

Pruebas Unitarias:

src > UnitTests.java > UnitTests

```java
import static org.junit.Assert.*;
import org.junit.Test;
import java.util.EmptyStackException;
import java.util.NoSuchElementException;
import java.util.ArrayList;

public class UnitTests {

    private static Calculadora calculadora = new Calculadora();
    private ArrayListStack<Integer> pila = new ArrayListStack();
    private ListaStack<Integer> listaStack = new ListaStack<>();
    private ListaSimple<Integer> listaSimple = new ListaSimple<>();
    private ListaDoble<Integer> listaDoble = new ListaDoble<>();

    @Test
    public void testEsDigito() {
        assertTrue(Calculadora.esDigito(caracter:'5'));
        assertFalse(Calculadora.esDigito(caracter:'+'));
    }

    @Test
    public void testSeparacion() {
        ArrayList<String> resultado = Calculadora.Separacion(texto:"3+5");
        assertEquals(3, resultado.size());
        assertEquals("3", resultado.get(index:0));
        assertEquals("+", resultado.get(index:1));
        assertEquals("5", resultado.get(index:2));
    }
```

```java
    @Test
    public void testCalcular() {
        String postfix = "34+";
        String seleccion = "VECTOR";
        String tipoADT = "STACK";

        int resultado = calculadora.calcular(postfix, seleccion, tipoADT);
        assertEquals(7, resultado);
    }

    @Test
    public void testCalcularConLista() {
        String postfix = "35+";
        String tipoADT = "LISTA"; // Ajustar según la implementación real
        String seleccion = "LISTA_SIMPLE";

        int resultado = Calculadora.calcular(postfix, seleccion, tipoADT);
        System.out.println(resultado);
        assertEquals(8, resultado);
    }
```

```java

    @Test
    public void testArrayListStack() {
        // Verifica si la pila está vacía al inicio
        assertTrue(pila.isEmpty());

        // Agrega elementos a la pila y verifica si la pila ya no está vacía
        pila.push(elemento:5);
        assertFalse(pila.isEmpty());

        // Verifica que el elemento en la parte superior de la pila sea el último agregado
        assertEquals(Integer.valueOf(i:5), pila.top());

        // Agrega más elementos a la pila
        pila.push(elemento:10);
        pila.push(elemento:15);

        // Verifica que al hacer pop, los elementos se retiren correctamente y en el orden adecuado
        assertEquals(Integer.valueOf(i:15), pila.pop());
        assertEquals(Integer.valueOf(i:10), pila.pop());
        assertEquals(Integer.valueOf(i:5), pila.pop());

        // Verifica que la pila esté vacía nuevamente después de hacer pop en todos los elementos
        assertTrue(pila.isEmpty());

        // Verifica que lanzar una excepción al hacer pop en una pila vacía
        try {
            pila.pop();
            fail("Se esperaba una IllegalStateException al hacer pop en una pila vacía");
        } catch (IllegalStateException e) {
            // Excepción esperada
        }

        // Verifica que lanzar una excepción al hacer top en una pila vacía
        try {
            pila.top();
            fail("Se esperaba una IllegalStateException al hacer top en una pila vacía");
        } catch (IllegalStateException e) {
            // Excepción esperada
        }
    }
```

```java

    @Test
    public void testListaStack() {
        // Verifica si la pila está vacía al inicio
        assertTrue(listaStack.isEmpty());

        // Agrega elementos a la pila y verifica si la pila ya no está vacía
        listaStack.push(elemento:5);
        assertFalse(listaStack.isEmpty());

        // Verifica que el elemento en la parte superior de la pila sea el último agregado
        assertEquals(Integer.valueOf(i:5), listaStack.top());

        // Agrega más elementos a la pila
        listaStack.push(elemento:10);
        listaStack.push(elemento:15);

        // Verifica que al hacer pop, los elementos se retiren correctamente y en el orden adecuado
        assertEquals(Integer.valueOf(i:15), listaStack.pop());
        assertEquals(Integer.valueOf(i:10), listaStack.pop());
        assertEquals(Integer.valueOf(i:5), listaStack.pop());

        // Verifica que la pila esté vacía nuevamente después de hacer pop en todos los elementos
        assertTrue(listaStack.isEmpty());

        // Verifica que lanzar una excepción al hacer pop en una pila vacía
        try {
            listaStack.pop();
            fail("Se esperaba una IllegalStateException al hacer pop en una pila vacía");
        } catch (IllegalStateException e) {
            // Excepción esperada
        }

        // Verifica que lanzar una excepción al hacer top en una pila vacía
        try {
            listaStack.top();
            fail("Se esperaba una IllegalStateException al hacer top en una pila vacía");
        } catch (IllegalStateException e) {
            // Excepción esperada
        }
    }

```

```java

    @Test
    public void testListaSimple() {
        // Verifica si la lista está vacía al inicio
        assertTrue(listaSimple.isEmpty());

        // Agrega elementos a la lista y verifica si la lista ya no está vacía
        listaSimple.push(elemento:5);
        assertFalse(listaSimple.isEmpty());

        // Verifica que el elemento en la parte superior de la lista sea el último agregado
        assertEquals(Integer.valueOf(i:5), listaSimple.top());

        // Agrega más elementos a la lista
        listaSimple.push(elemento:10);
        listaSimple.push(elemento:15);

        // Verifica que al hacer pop, los elementos se retiren correctamente y en el orden adecuado
        assertEquals(Integer.valueOf(i:15), listaSimple.pop());
        assertEquals(Integer.valueOf(i:10), listaSimple.pop());
        assertEquals(Integer.valueOf(i:5), listaSimple.pop());

        // Verifica que la lista esté vacía nuevamente después de hacer pop en todos los elementos
        assertTrue(listaSimple.isEmpty());

        // Verifica que lanzar una excepción al hacer pop en una lista vacía
        try {
            listaSimple.pop();
            fail("Se esperaba una IllegalStateException al hacer pop en una lista vacía");
        } catch (IllegalStateException e) {
            // Excepción esperada
        }

        // Verifica que lanzar una excepción al hacer top en una lista vacía
        try {
            listaSimple.top();
            fail("Se esperaba una IllegalStateException al hacer top en una lista vacía");
        } catch (IllegalStateException e) {
            // Excepción esperada
        }
    }

```

```java

    @Test
    public void testListaDoble() {
        // Verifica si la lista está vacía al inicio
        assertTrue(listaDoble.isEmpty());

        // Agrega elementos a la lista y verifica si la lista ya no está vacía
        listaDoble.push(elemento:5);
        assertFalse(listaDoble.isEmpty());

        // Verifica que el elemento en la parte superior de la lista sea el último agregado
        assertEquals(Integer.valueOf(i:5), listaDoble.top());

        // Agrega más elementos a la lista
        listaDoble.push(elemento:10);
        listaDoble.push(elemento:15);

        // Verifica que al hacer pop, los elementos se retiren correctamente y en el orden adecuado
        assertEquals(Integer.valueOf(i:15), listaDoble.pop());
        assertEquals(Integer.valueOf(i:10), listaDoble.pop());
        assertEquals(Integer.valueOf(i:5), listaDoble.pop());

        // Verifica que la lista esté vacía nuevamente después de hacer pop en todos los elementos
        assertTrue(listaDoble.isEmpty());

        // Verifica que lanzar una excepción al hacer pop en una lista vacía
        try {
            listaDoble.pop();
            fail("Se esperaba una IllegalStateException al hacer pop en una lista vacía");
        } catch (IllegalStateException e) {
            // Excepción esperada
        }

        // Verifica que lanzar una excepción al hacer top en una lista vacía
        try {
            listaDoble.top();
            fail("Se esperaba una IllegalStateException al hacer top en una lista vacía");
        } catch (IllegalStateException e) {
            // Excepción esperada
        }
    }
}

```

Ventajas y Desventajas de Singleton:

Ventajas:

- Permite hacer uso de una única instancia en toda la aplicación.
- Se tiene acceso global a la instancia, facilita su acceso desde cualquier punto.
- Se mejora el uso de memoria al evitar estar creando objetos constantemente.
- Aumenta la eficiencia, al solo tener que crear el objeto una vez.

Desventajas:

- Al ser global, puede presentar algunos problemas al estar haciendo pruebas unitarias.
- Se tiene que tener cuidado con su implementación, pues, se pueden presentar casos en los que necesitemos varios objetos, cada uno con información única, y el Singlenton lo impediría.

¿Fue adecuado su uso para el programa?

- El equipo considera que sí lo fue, pues al ser una variable global, nos permitía realizar cálculos desde cualquier punto sin necesidad de tener que generar una instancia de manera constante, esto mejoraba el rendimiento del programa y su eficiencia.
- Al ser un objeto que no se utilizaba para almacenar información distintiva, sino que solo hacía cálculos, aplicarle el patrón de diseño de Singlenton quedaba perfecto, pues con tener una calculadora global se podían llevar todos los procesos a cabo.