

Hoja de trabajo 4

• Ventajas y desventajas del patrón de diseño Singleton

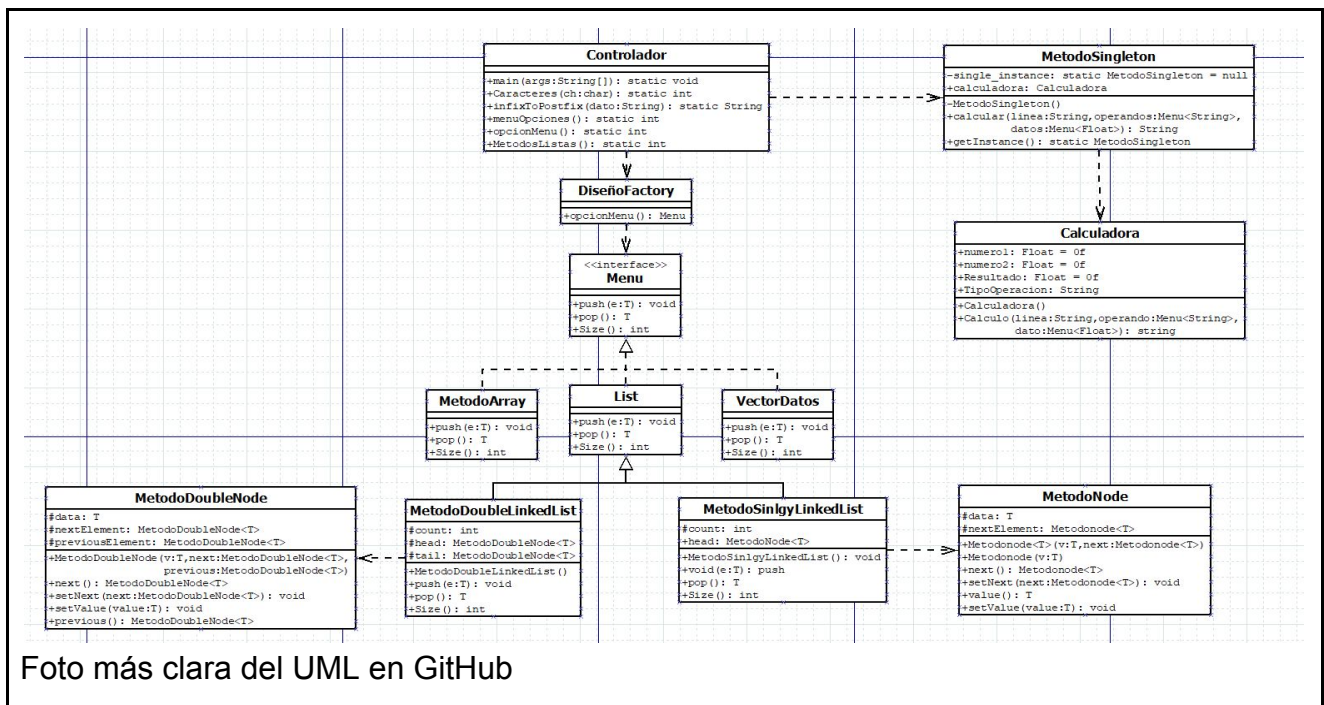
- Algunas ventajas del patrón de diseño Singleton son que no utiliza memoria hasta que es llamado, por lo que el diseño Singleton es útil para clases con recursos muy intensivos. Adicionalmente también puede ser expandido al diseño Factory, y también puede heredar de otras clases.
- Algunas de las desventajas del patrón de diseño Singleton son que hacen que los Unit tests sean más difíciles de realizar ya que crea un estado global. También, reduce el potencial de paralelismo en el programa.

• ¿Cree que su uso es adecuado en este programa?

- Nosotros pensamos que su uso si es adecuado en este programa porque en este tipo de programa hace su función de solo permitir que se cree una instancia de la calculadora ya que no se necesita nada más. Adicionalmente, como en este programa también había que implementar el patrón de diseño Factory, el uso de Singleton es favorable utilizarlo ya que se puede extenderlo al diseño Factory.
- Y con esto se podría decir que nos reduce bastante el trabajo extra como también la manera visual en la que se terminaría viendo nuestro código. Que a su vez con muy pocas líneas, el uso de singleton permite que se realice una sola instancia de un solo uso. Cosa que al momento de aplicarla al diseño Factory, nos ahorra bastante trabajo a la hora de correrlo como también de crear las instancias.

• Diagrama de clases

Versión 1 UML:



- Unit Tests
 - Lista

```

1 import java.util.LinkedList;
2
3 import static org.junit.jupiter.api.Assertions.*;
4
5 class ListTest<T> {
6
7     @org.junit.jupiter.api.Test
8     void push() {
9         List<T> l = new List<T>();
10        l.datos.push((T) "hello");
11        assertEquals( expected: "hello", l.datos.getFirst());
12    }
13
14     @org.junit.jupiter.api.Test
15     void pop() {
16         List<T> l = new List<T>();
17         l.datos.push((T) "hello");
18         l.datos.pop();
19         int longitud = l.datos.size();
20         assertEquals( expected: 0, longitud);
21     }
22
23     @org.junit.jupiter.api.Test
24     void size() {
25         List<T> l = new List<T>();
26         l.datos.push((T) "hello");
27         int longitud = l.datos.size();
28         assertEquals( expected: 1, longitud);
29     }
30 }

```

Run: ListTest ×

Tests passed: 3 of 3 tests - 17 ms

Test Results	Time	Process
Test Results	17 ms	"C:\Program Files\Java\jdk-14.0.1\bin\java.exe" ...
✓ ListTest	17 ms	
✓ pop()	14 ms	
✓ push()	3 ms	
✓ size()	3 ms	

Process finished with exit code 0

- Pila

```

1 import ...
2
3 class VectorDatosTest<T> {
4
5     @Test
6     void push() {
7         List<T> l = new List<T>();
8         l.datos.push((T) "Hola");
9         assertEquals( expected: "Hola", l.datos.getFirst());
10    }
11
12     @Test
13     void pop() {
14         List<T> l = new List<T>();
15         l.datos.push((T) "Hola");
16         l.datos.pop();
17         int longitud = l.datos.size();
18         assertEquals( expected: 0, longitud);
19     }
20
21     @Test
22     void size() {
23         List<T> l = new List<T>();
24         l.datos.push((T) "Hola");
25         int longitud = l.datos.size();
26         assertEquals( expected: 1, longitud);
27     }
28 }

```

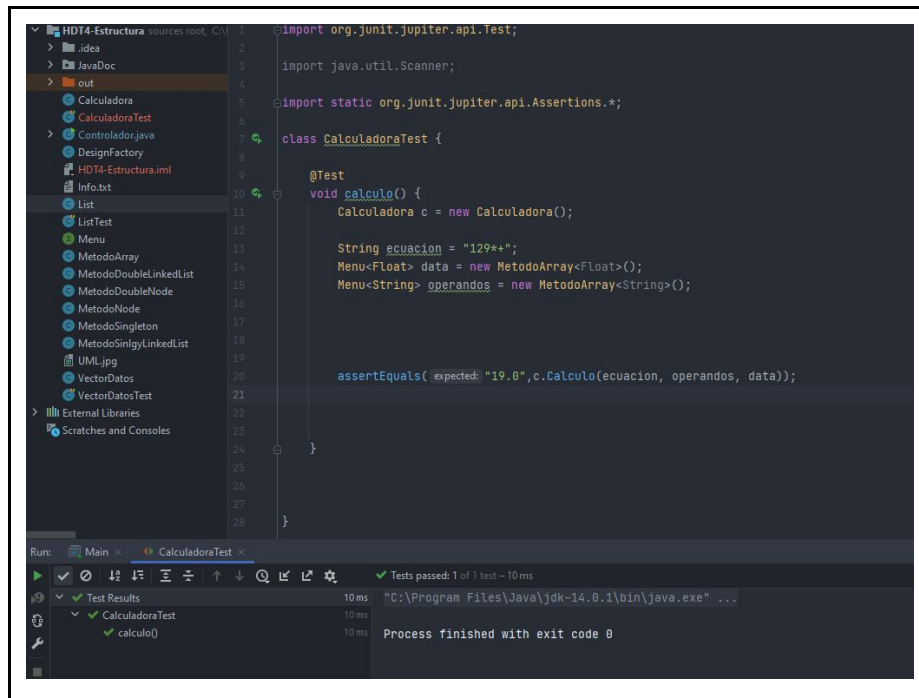
Run: VectorDatosTest ×

Tests passed: 3 of 3 tests - 16 ms

Test Results	Time	Process
Test Results	16 ms	"C:\Program Files\Java\jdk-14.0.1\bin\java.exe" ...
✓ VectorDatosTest	16 ms	
✓ pop()	12 ms	
✓ push()	2 ms	
✓ size()	2 ms	

Process finished with exit code 0

- **Calculadora**



- **GitHub**

- <https://github.com/bar20807/HDT4-Estructura>