Autores: María José Villafuerte 22129	**Docente:** Moises Alonso
Sección: 10	**Fecha:** 20/02/2023

Hoja de trabajo # 4

Algoritmos de Ordenamiento

1. Repositorio utilizado:

a. https://github.com/Maria-Villafuerte/Hoja_de_trabajo_4.1 Inicialmente era este pero se corrompió por que instalé una librería que no debía y me hizo estragos en el programa

b. https://github.com/Maria-Villafuerte/Hoja_de_Trabajo_4.2

2. Investigación

Revise que ventajas / desventajas hay al utilizar el patrón Singlenton en general:

Ventajas:

- Garantiza que solo exista una instancia de la clase
- Proporciona un punto de acceso global a la instancia de la clase, lo que facilita su uso en diferentes partes del código.
- Permite la definición de un control de acceso centralizado para la instancia única de la clase, lo que puede ser útil para la sincronización en hilos de ejecución concurrentes.
- Ayuda a reducir el uso de memoria y mejora el rendimiento.
- Facilita el acceso a la funcionalidad que ofrece la clas

Desventajas:

- Puede dificultar el testeo de la clase que implementa el patrón Singleton, ya que puede ser complicado crear una instancia falsa o simulada de la clase.
- Puede hacer que el código sea menos flexible, ya que la clase se convierte en un recurso compartido global que puede ser difícil de cambiar.
- Puede introducir problemas de rendimiento si se utiliza mal, ya que la instancia única de la clase puede requerir mucha memoria y recursos de procesamiento.

¿Cree que su uso es adecuado en este programa?

Yo creo que el patrón Factory si es sumamente importante para hacer cambio de stacks sin necesidad de hacer diferentes métodos, de otra forma considero que el patrón singletone no va mucho con el programa solicitado ya que no veo la necesidad de tener una clases que solo se pueda llamar a ella misma.

3. Pruebas Unitarias JUnit

```
class CalculadoraTest {
    PostfixCalculator pc = new PostfixCalculator();
    Stack s = new Stack();

    CalculadoraTest() {
        this.s.push( value: 1);
        this.s.push( value: 3);
    }

    @Test
    void peek_Test() { Assertions.assertEquals(3, this.s.peek()); }

    @Test
    void count_Test() { Assertions.assertEquals(2, this.s.count()); }

    @Test
    void getItems_Test() {
        ArrayList<String> temp = new ArrayList();
        temp.add("1");
        temp.add("2");
        temp.add("+");
        temp.add("4");
        temp.add("*");
        temp.add("3");
        temp.add("+");
        Assertions.assertEquals(temp, this.pc.getItems( _expresion: "1 2 + 4 * 3 + "));
    }
```