**Universidad de Las Américas**

Facultad de Ingenierías y Ciencias Agropecuarias

*Ingeniería de Softwate*

**Informe de laboratorio**

1. **DATOS DEL ALUMNO: Dylan Clerque**
2. **TEMA DE LA PRÁCTICA: Patrón Singleton**
3. **OBJETIVO DE LA PRÁCITCA**

Garantizar que solo exista una única instancia de una clase en todo el programa. Esto asegura que no haya instancias duplicadas y que se pueda acceder de manera centralizada a los datos y métodos de la instancia.

1. **OBJETIVOS ESPECÍFICOS**

El patrón Singleton ofrece un punto de acceso global a la instancia única de la clase, lo que permite que diferentes partes del programa accedan a ella de manera fácil y rápida.

Al centralizar la creación y acceso a la instancia de la clase, se puede controlar quién accede a la instancia y cuándo.

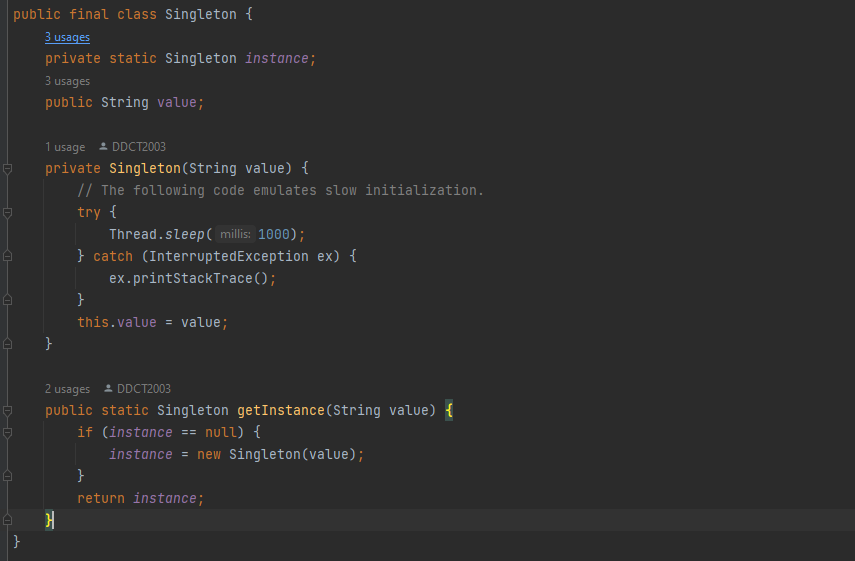
Al tener una única instancia, se evita la creación innecesaria de múltiples objetos de la misma clase, lo que puede ahorrar recursos de memoria y procesamiento.

1. **MATERIALES Y MÉTODOS**

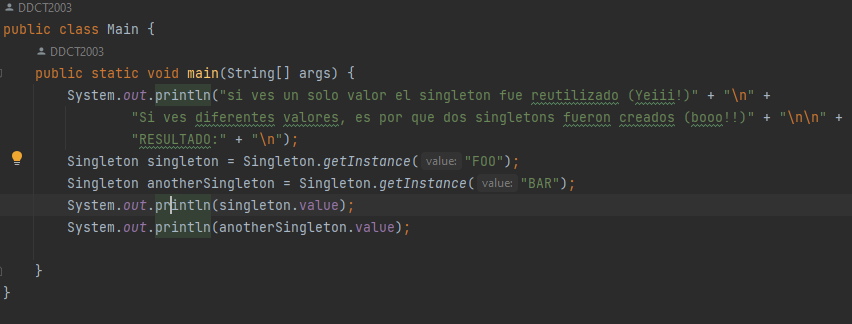
* **IntellIJ**
* **Patrón de diseño singleton**

1. **DESARROLLO DE LA PRÁCTICA Y RESULTADOS**

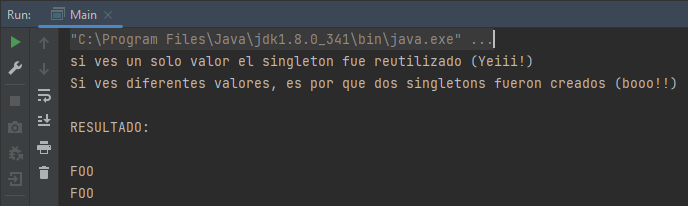
**Clase Singleton:**

****

**Main:**

****

**Resultado:**

****

1. **OPINIÓN PERSONAL**

**Aquí tienes 4 opiniones sobre el patrón Singleton:**

1. El patrón Singleton puede ser muy útil en situaciones donde se necesita administrar eficientemente los recursos del sistema, como en la gestión de conexiones a bases de datos, registros de auditoría o configuraciones. Tener una única instancia evita el desperdicio de recursos y la duplicación de esfuerzos.

2. Una crítica común al patrón Singleton es que puede dificultar las pruebas unitarias, ya que puede introducir dependencias globales y estados compartidos. Esto puede complicar la simulación de diferentes estados en las pruebas y aumentar el acoplamiento entre los componentes.

3. Al centralizar toda la lógica de una clase en una única instancia, existe el riesgo de crear un punto único de fallo en la aplicación. Si la instancia falla o presenta errores, puede afectar a todo el sistema.

4. Aunque el patrón Singleton tiene sus beneficios, es importante usarlo con moderación. Puede ser innecesario o incluso perjudicial en algunos casos, como en sistemas altamente concurridos donde se requiere un alto grado de paralelismo y escalabilidad. Además, algunos desarrolladores pueden sobreutilizar el patrón Singleton incluso cuando no es necesario, lo que puede conducir a problemas de diseño y mantenimiento a largo plazo.

1. **ANEXOS (Si se requiere)**

Github: <https://github.com/DDCT2003/Singleton-patr-n>

1. **BIBLIOGRAFÍA (Si se requiere)**

*Singleton en Java / Patrones de diseño*. (n.d.). <https://refactoring.guru/es/design-> patterns/singleton/java/example