



PATRÓN SINGLETON



Mauricio Delgado

Javier Pupo



AGENDA



- 1 **Introducción**
- 2 **Descripción del patrón**
- 3 **Implementaciones**
- 4 **Ventajas y Desventajas**
- 5 **Ejemplos de Uso**
- 6 **Relaciones con otros patrones**



2

¿QUÉ ES EL PATRÓN
SINGLETON?

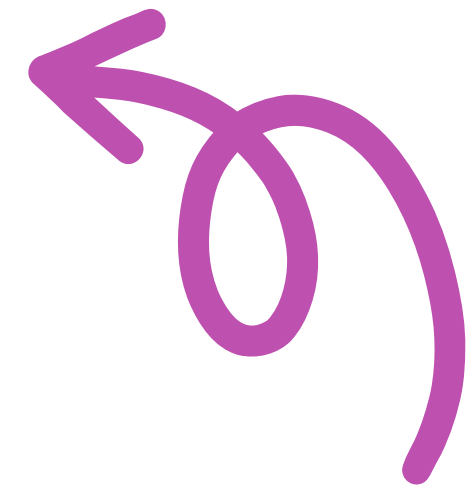
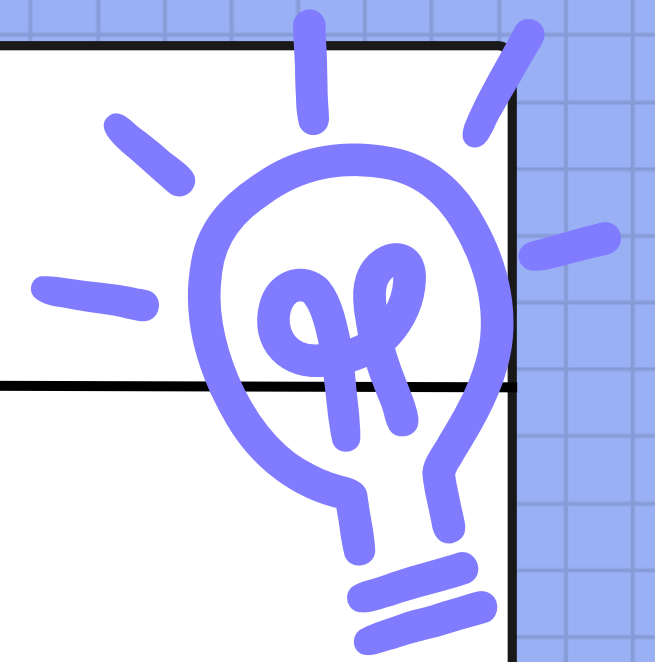
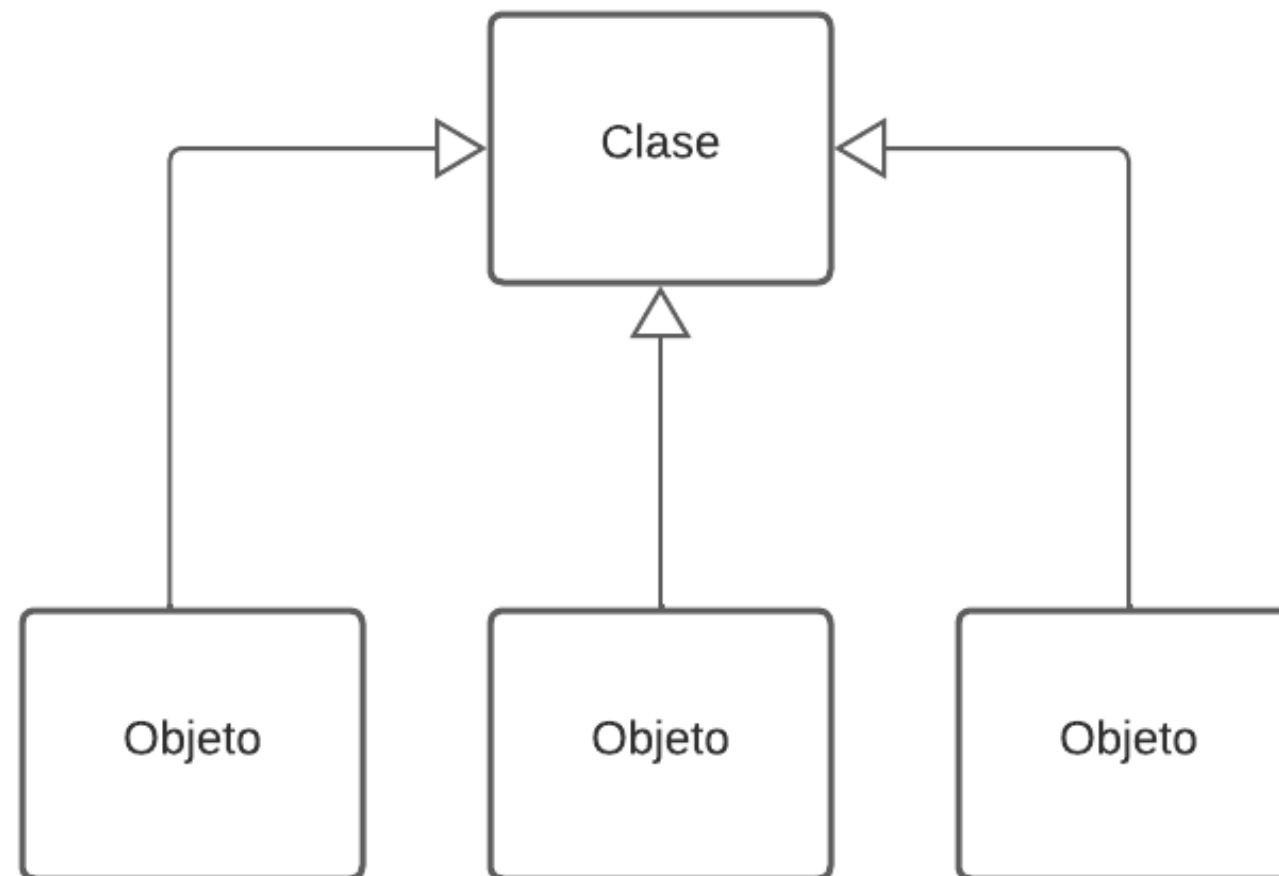


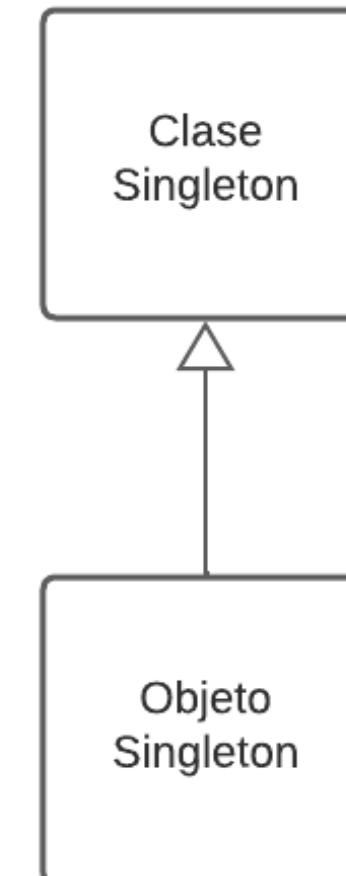
DIAGRAMA SINGLETON



Convencional

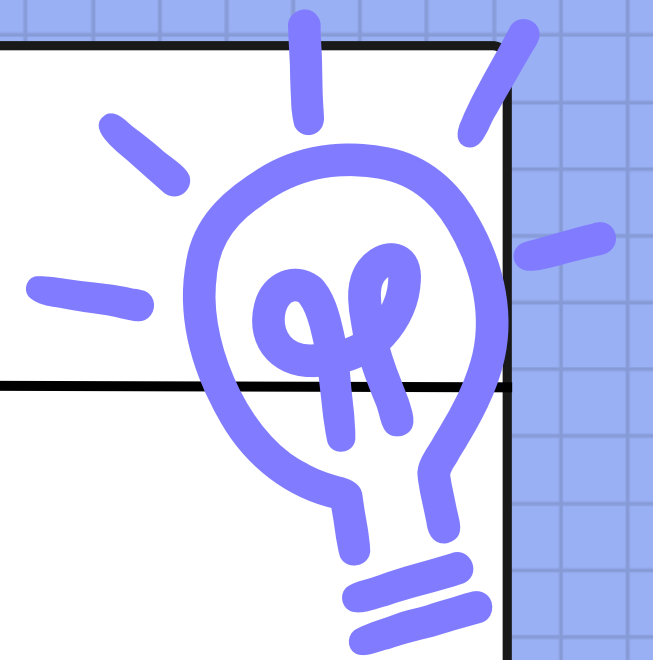


Singleton





PROPIEDADES



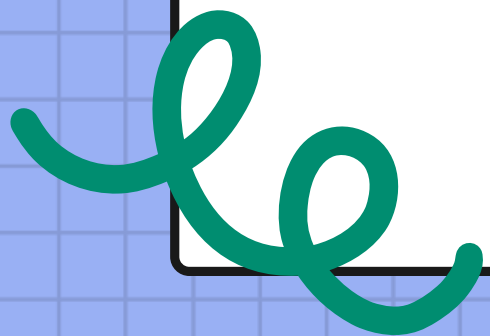
- **INSTANCIA ÚNICA**

Se logra creando una única instancia desde dentro de la clase. Se evita que las subclases o clases externas tengan acceso al constructor

- **INSTANCIA PÚBLICA**

En lenguajes con control de accesibilidad se debe crear una instancia pública.

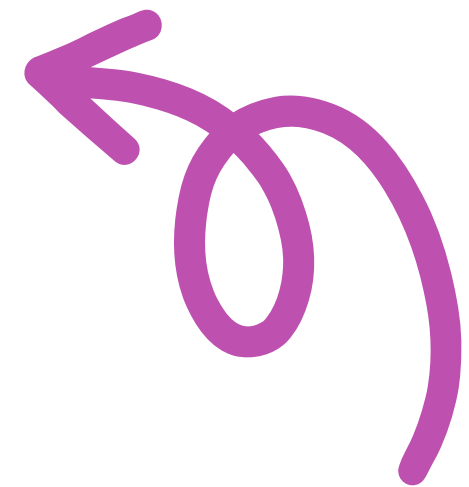
```
public class Singleton {  
  
    //Instancia pública accesible globalmente  
    public static singleton = new Singleton();  
  
    private Singleton(){  
        //Constructor privado  
    }  
}
```

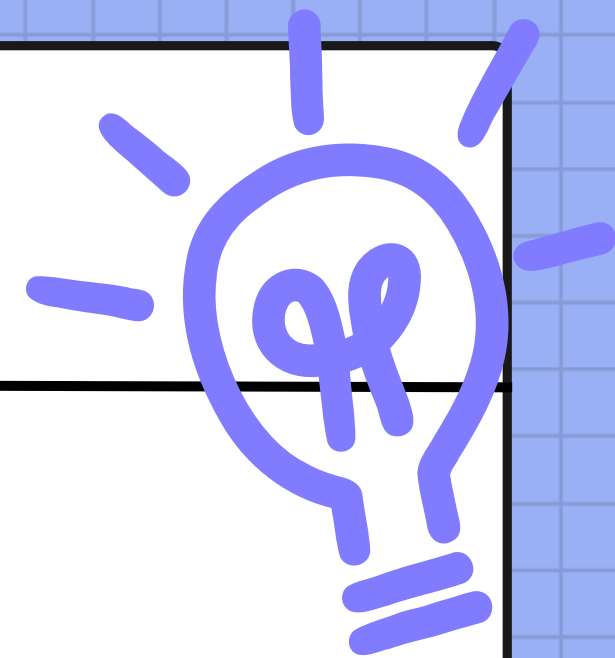




③

IMPLEMENTACIÓN





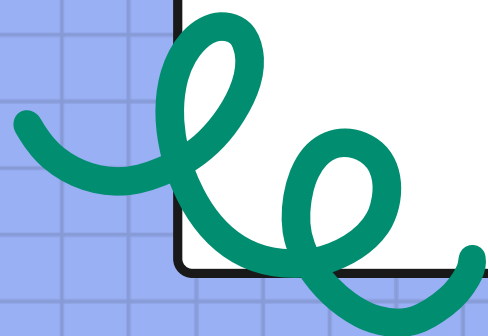
MANERAS DE INICIALIZAR

- **EARLY INITIALIZATION**

Se inicializa la clase aunque no se use la instancia.

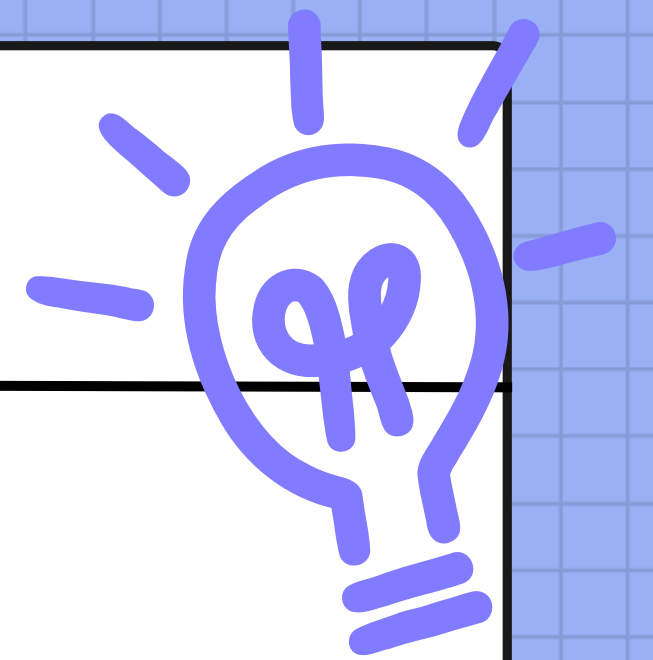
- **LAZY INITIALIZATION**

No se inicializa la instancia hasta que sea necesaria.
Es la más común





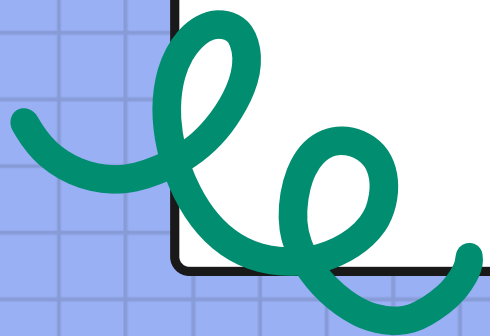
IMPLEMENTACIÓN CLÁSICA



```
class Singleton
{
    private static Singleton obj;

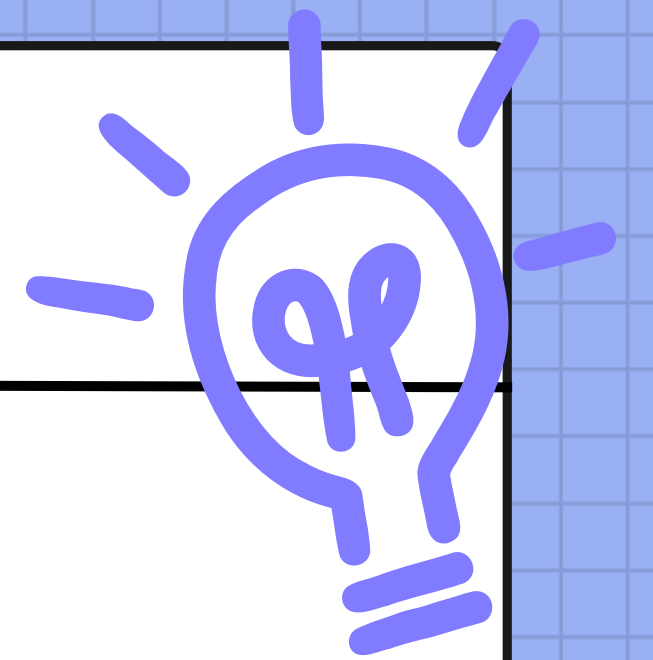
    //Constructor privado
    private Singleton() {}

    public static Singleton getInstance()
    {
        if (obj==null)
        {
            obj = new Singleton();
        }
        return obj;
    }
}
```





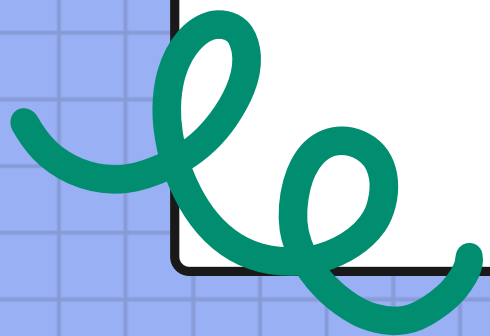
THREAD SAFE



```
class Singleton
{
    private static Singleton obj;

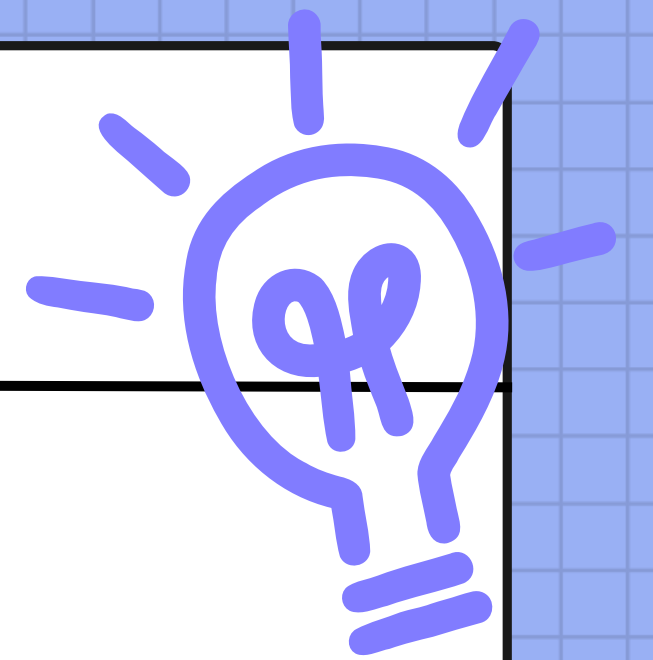
    private Singleton() {}

    // Only one thread can execute this at a time
    public static synchronized Singleton getInstance()
    {
        if (obj==null)
            obj = new Singleton();
        return obj;
    }
}
```





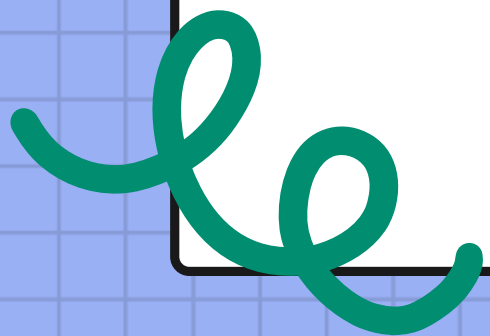
EAGER/IMPACIENTE



```
class Singleton
{
    private static Singleton obj = new Singleton();

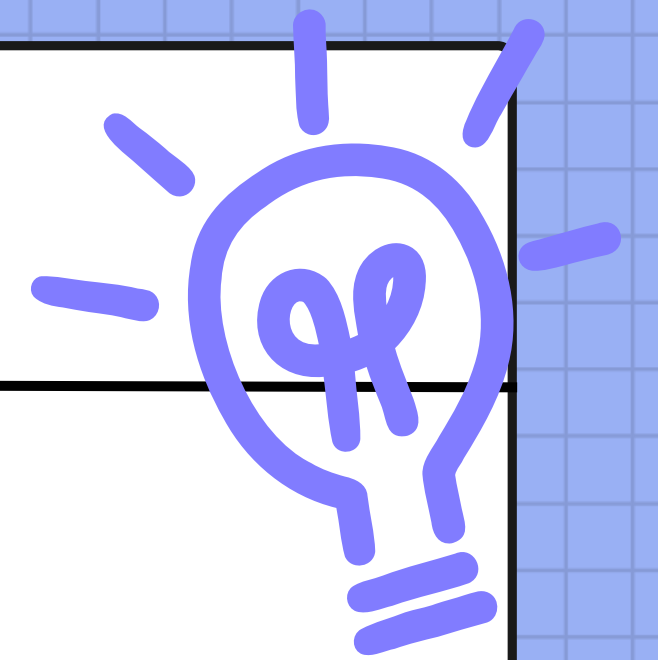
    private Singleton() {}

    public static Singleton getInstance()
    {
        return obj;
    }
}
```





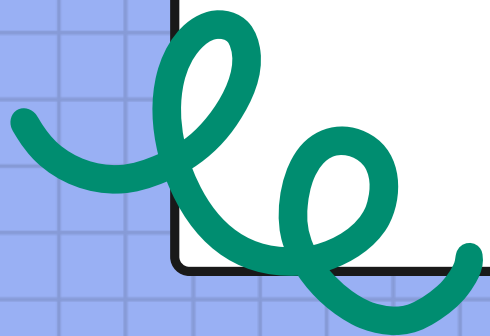
DOUBLE CHECKED LOCKING



```
class Singleton
{
    private static volatile Singleton obj = null;

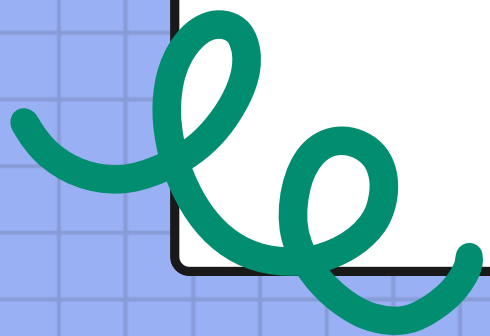
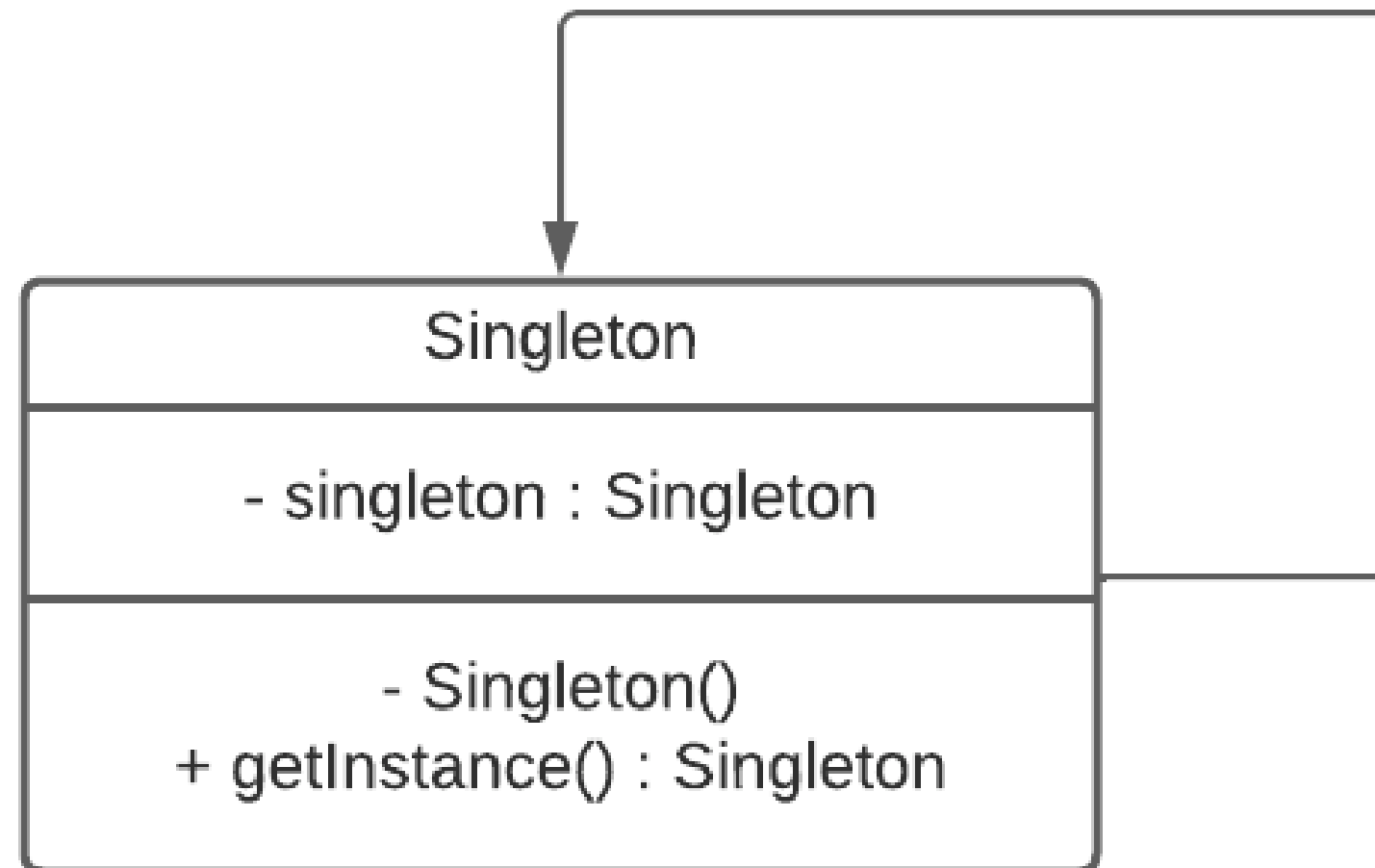
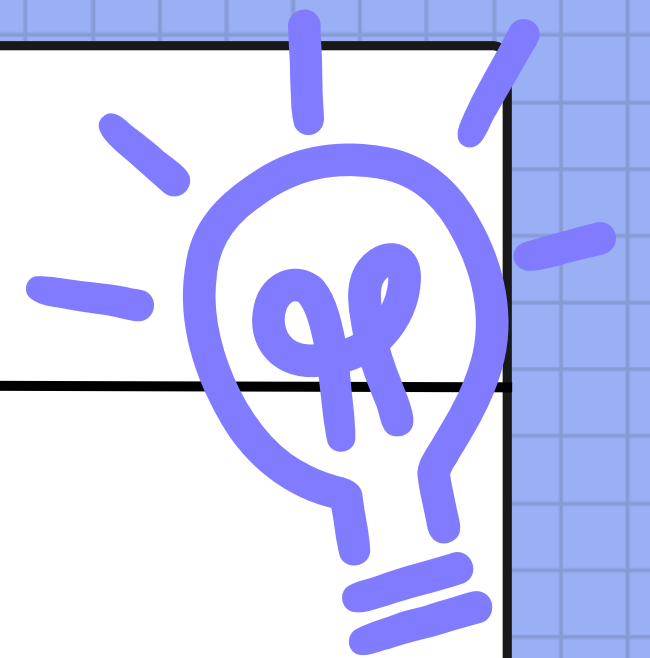
    private Singleton() {}

    public static Singleton getInstance()
    {
        if (obj == null)
        {
            // thread safe
            synchronized (Singleton.class)
            {
                // En caso de que varios
                // hilos lleguen acá
                if (obj==null)
                {
                    obj = new Singleton();
                }
            }
        }
        return obj;
    }
}
```





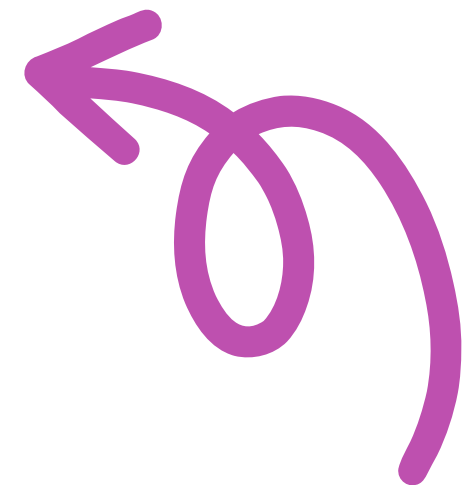
CLASE SINGLETON






4

VENTAJAS Y DESVENTAJAS





Ventajas

- Control de acceso concurrente
 - Un objeto singleton se instancia una única vez
 - Objeto disponible en un estado controlado
 - Puede implementar interfaces
 - Fácil de implementar
- 

Desventajas



- No es thread safe
- Resuelve dos problemas a la vez. No sigue el Single Responsibility Principle
- Su estado siempre activo complica unit testing



EJEMPLOS





EJEMPLO IMPLEMENTACION

```
#include <iostream>

class Singleton {
private:
    // Variable estática privada para contener la única instancia de la clase
    static Singleton* instance;

    // Constructor privado para evitar la creación directa de objetos.
    Singleton() {
        std::cout << "Creando una instancia singleton " << std::endl;
    }

public:
    // Método estático público para obtener la única instancia de la clase.
    static Singleton* get_instance() {
        // Comprobar si ya se ha creado una instancia de la clase
        if (instance == nullptr) {
            // Si no, crea una nueva instancia de la clase.
            instance = new Singleton();
        }
        return instance;
    }

    // Método público para demostrar el uso de la instancia singleton
    void demonstrate() {
        std::cout << "Demostración del uso de la instancia singleton " << std::endl;
    }
};

// Inicializar variable de instancia estática a nula
Singleton* Singleton::instance = nullptr;

int main() {
    // Obtener la instancia singleton
    Singleton* my_singleton = Singleton::get_instance();

    // Demostrar el uso de la instancia singleton
    my_singleton->demonstrate();

    return 0;
}
```




CONEXIÓN A BASE DE DATOS

```
#include <mysql.h> // Incluimos la librería de MySQL

class Database {
private:
    static Database* instance; // Variable estática para almacenar la única instancia de la clase
    MYSQL* connection;

    Database() { // Constructor privado para evitar que se creen instancias desde fuera de la clase
        connection = mysql_init(NULL); // Inicializamos la conexión
        if (!mysql_real_connect(connection, "localhost", "user", "password", "DB", 0, NULL, 0)) {
            std::cout << "Error al conectar con la base de datos: " << mysql_error(connection) << std::endl;
        }
    }

public:
    static Database* getInstance() { // Método estático para obtener la única instancia de la clase
        if (!instance) {
            instance = new Database();
        }
        return instance; // Devolvemos la instancia creada
    }

    MYSQL* getConnection() { return connection; }

    ~Database() { mysql_close(connection); }
};
```



CONEXIÓN A BASE DE DATOS

```
Database* Database::instance = nullptr; // Inicializamos la variable estática a nullptr

int main() {
    Database* db = Database::getInstance();
    MYSQL* conn = db->getConnection();
    // Aquí podemos utilizar la conexión para realizar operaciones en la base de datos
    return 0;
}
```



BUILDER SINGLETON

```
class Product {
public:
    void setPart1(const std::string& part1);
    void setPart2(const std::string& part2);
    std::string getParts() const;
private:
    std::string part1_;
    std::string part2_;
};

class ProductBuilder {
public:
    ProductBuilder& setPart1(const std::string& part1);
    ProductBuilder& setPart2(const std::string& part2);
    Product build() const {
        return product_;
    }
private:
    Product product_;
};
```



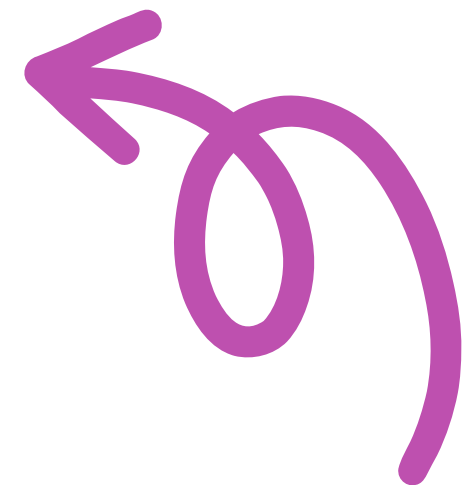
BUILDER SINGLETON

```
class ProductBuilderFactory {
public:
    static ProductBuilder& getInstance() {
        static ProductBuilder builder;
        return builder;
    }
};

int main() {
    // Utilizamos la instancia del Builder Singleton para crear un objeto de tipo Product
    Product product = ProductBuilderFactory::getInstance().setPart1("Parte 1").setPart2("Parte 2").build();
    std::cout << product.getParts() << std::endl; // Imprime "Parte 1 Parte 2"
    return 0;
}
```

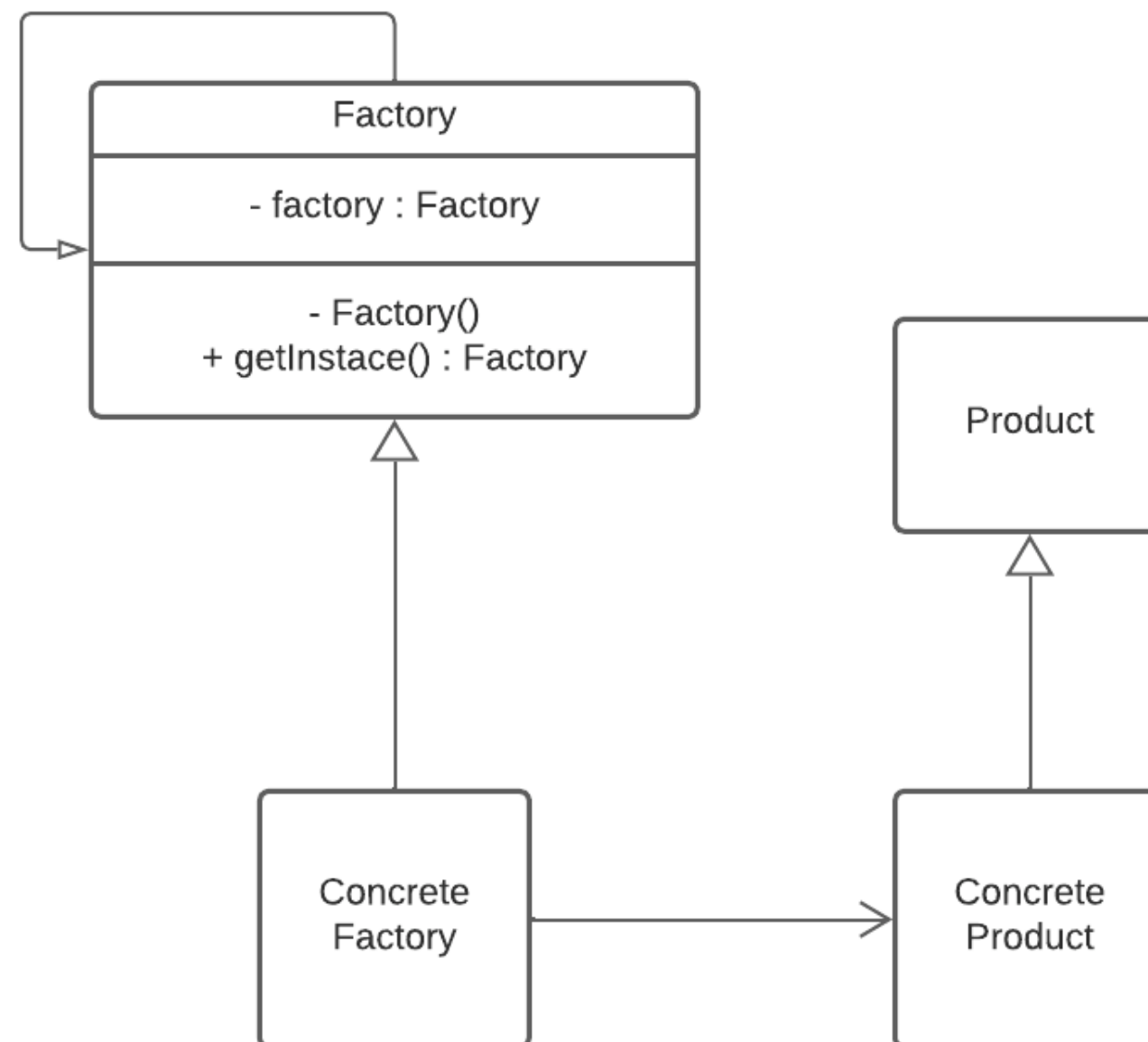
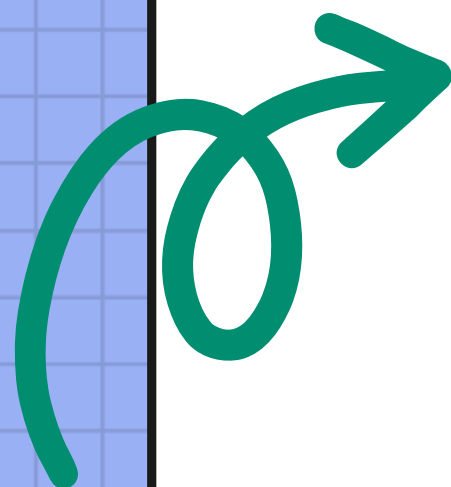
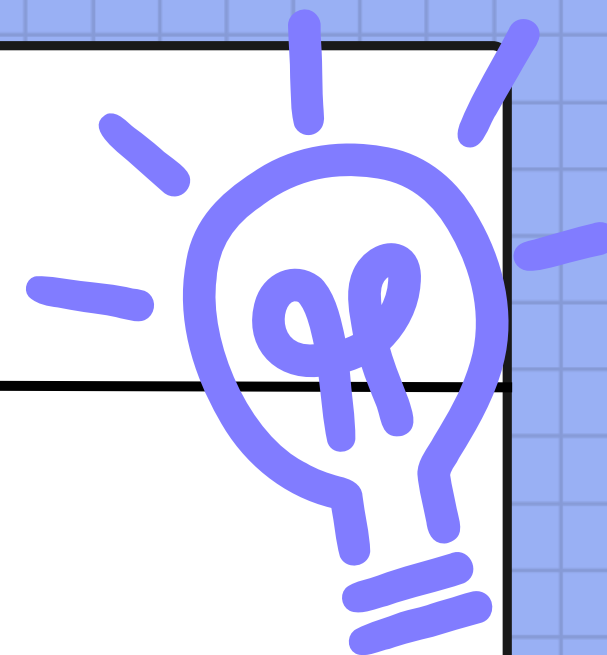


RELACIÓN CON OTROS PATRONES





FACTORY METHOD





PATRONES RELACIONADOS



BUILDER*



ADAPTER

DECORATOR

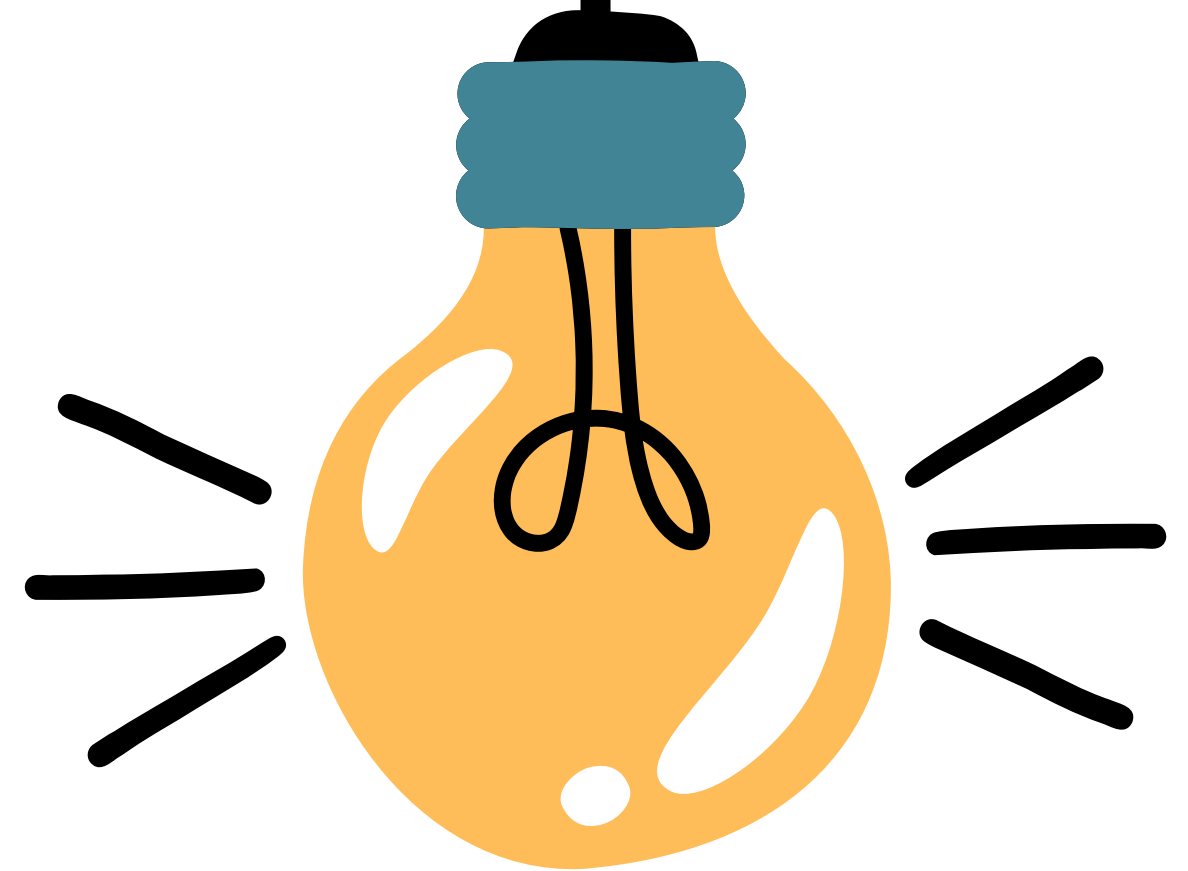


PROXY

OBSERVER



**MUCHAS
GRACIAS**





BIBLIOGRAFÍA



The Singleton — Python 3 Patterns, Recipes and Idioms. (n.d.). <https://python-3-patterns-idioms-test.readthedocs.io/en/latest/Singleton.html>

GeeksforGeeks. (2021). Singleton Design Pattern Introduction. *GeeksforGeeks*.
<https://www.geeksforgeeks.org/singleton-design-pattern-introduction/?ref=gcse>

Singleton. (n.d.). <https://refactoring.guru/es/design-patterns/singleton>

GeeksforGeeks. (2020). Singleton Design Pattern Implementation. *GeeksforGeeks*.
<https://www.geeksforgeeks.org/singleton-design-pattern/?ref=rp>




Actividad

En un aula mística, fuertemente influenciada por patrones y diseños, los lunes y jueves se sientan a escuchar las historias de los sabios computines. Hoy van a contar la leyenda de Singleton.

Cuenta la leyenda que una vez existió Singleton, un objeto único y disponible para quien lo pidiera. Este Singleton garantizaba graduación con honores a quien lo tuviera en su poder, Pero su costo era el orden y la paz. Las personas estaban dispuestas a hacer lo que fuera para tener a Singleton en su poder.

Con lo que no contaban era que, justo hoy, Singleton se le iba a presentar a una pareja.



Turnos

1

Noche

- Todo el pueblo está dormido
- Singleton hace su aparición

2

Día

- El aula destierra una pareja
- Esperan al día siguiente

¿Cómo ganar?

- Ser la última persona con el Singleton