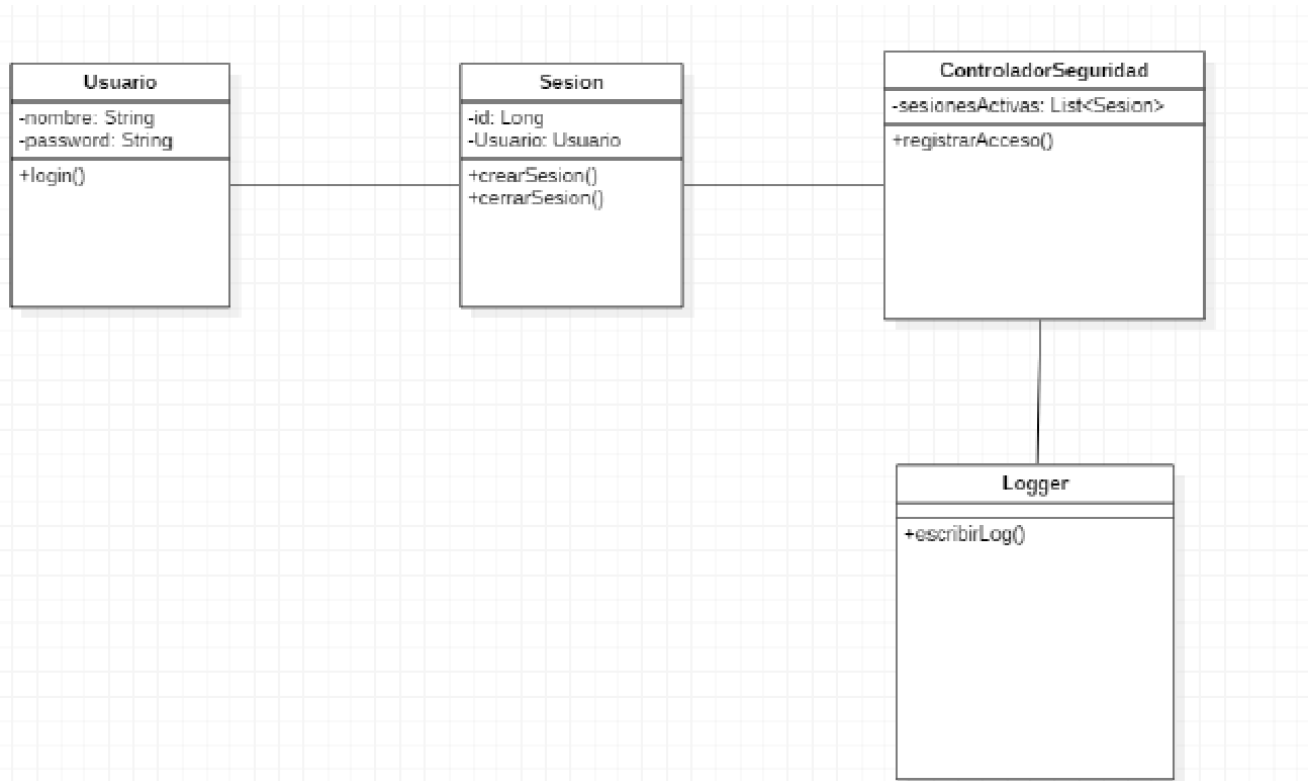


Guia N°1 de Patrones - Patrones creacionales

Ejercicio 1:

Considere el siguiente diagrama de UML, el cual modela un módulo de seguridad de una aplicación:



Dada ésta implementación se necesita:

- Se debe poder crear sólo una instancia de la clase *ControladorSeguridad* ya que existe un único objeto encargado de realizar dicha tarea.
- Cada vez que se registre el acceso de un usuario, la clase *Logger* debe escribir en un archivo el suceso, con el fin de registrar dicha información.

En base a esto se pide:

- Indicar que patrón / patrones se pueden utilizar para cada caso y justificar su uso.
- Reescribir el diagrama de UML con la solución implementada.

Ejercicio 2

Dado el siguiente código identifique que tipo de patrón se implementa y realice el diagrama correspondiente.

```
class Inicio_Serie
{
protected:
    Inicio_Serie(const std::string value): value_(value)
    {
    }
    static Inicio_Serie* Inicio_Serie_;
    std::string value_;

public:
    Inicio_Serie(Inicio_Serie &other) = delete;
    void operator=(const Inicio_Serie &) = delete;
    static Inicio_Serie *GetInstance(const std::string& value);
    void SomeBusinessLogic()
    {
    }
    std::string value() const{
        return value_;
    }
};

Inicio_Serie* Inicio_Serie::Inicio_Serie_ = nullptr;;
Inicio_Serie *Inicio_Serie::GetInstance(const std::string& value)
{
    if(Inicio_Serie_==nullptr){
        Inicio_Serie_ = new Inicio_Serie(value);
    }
    return Inicio_Serie_;
}

void ThreadFoo(){
    std::this_thread::sleep_for(std::chrono::milliseconds(1000));
    Inicio_Serie* Inicio_Serie = Inicio_Serie::GetInstance("FOO");
    std::cout << Inicio_Serie->value() << "\n";
}

void ThreadBar(){
    std::this_thread::sleep_for(std::chrono::milliseconds(1000));
    Inicio_Serie* Inicio_Serie = Inicio_Serie::GetInstance("BAR");
    std::cout << Inicio_Serie->value() << "\n";
}

int main()
{
    std::cout <<"Si ves el mismo valor, entonces fue exitoso (¡genial!\n" <<
        " Si ves un valor diferente entonces esta incorrecto(booo!!)\n" <<
        "RESULT:\n";
    std::thread t1(ThreadFoo);
    std::thread t2(ThreadBar);
    t1.join();
    t2.join();

    return 0;
}
```

Ejercicio 3

Una empresa que desarrolla sistemas software de terminal de punto de venta, TPV solicita que se diseñe un diagrama de clases implementando patrones de diseño. Estos sistemas deben permitir la conexión a ordenadores remotos para procesar los pagos por tarjeta de crédito. Cada comercio podrá utilizar diferentes servicios externos de autorización de pago para distintos tipos de tarjeta (uno para VISA, otro para MasterCard, etc.) cada uno de los cuales tiene su propia API. El tipo de compañía de crédito se puede deducir del código de la tarjeta (si empieza por 5 es una MasterCard, por 4 una VISA, etc.). Cada comercio tiene un identificador dependiendo del tipo de tarjeta que debe enviarse con la solicitud de pago.

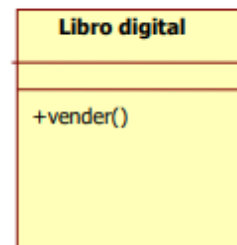
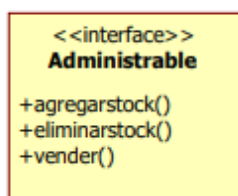
- Diseña una solución para que el sistema TPV pueda interactuar con diferentes sistemas externos de autorización de pago utilizando patrones de diseño.
- ¿Qué patrón utilizaste? ¿Qué ventaja tiene?

Ejercicio 4

Se quiere desarrollar una aplicación de gestión de stock de un negocio que vende diferentes familias de productos. Se necesita crear por ahora productos de música (CD,DVD) y productos de computación (pc, notebook, tablet). ¿Qué patrón utilizaría? ¿Cuál sería la estructura del mismo? Realizar el diagrama del patrón correspondiente.

Ejercicio 5

Supongamos que tenemos un sistema que vende libros. Se desea vincular al sistema una clase de tipo libro digital con un funcionamiento diferente al que se está manejando. Se debe adaptar la nueva clase sin que afecte la lógica de la aplicación. Se adjunta una parte del diagrama de UML:



Ejercicio 6

Diga qué patrón se está utilizando en este ejemplo e indique cuáles son las partes de la estructura del patrón.

