Documentación del proyecto

Propósito del programa: *Busca llevar a cabo la gestión y organización de un banco, dar de alta a clientes, eliminarlos y agregar cuenta a los clientes, de igual forma una manera de visualizar los cambios realizados para el fácil manejo del usuario*

Historias de usuario

Agregar y Eliminar Clientes
Como usuario
Quiero agregar y eliminar clientes de mi base de datos
Para mantener actualizado mis registros

Imprimir lista de Clientes
Como usuario
Quiero visualizar los cambios realizados a los clientes de mi base de datos
Para tener una idea de cómo se comportan los registros y acceder a ellos de manera más sencilla

Modificar lista de Clientes
Como usuario
Quiero realizar cambios a los clientes y a sus cuentas
Para eliminar sus cuentas si así es requerido

Patrones de diseño

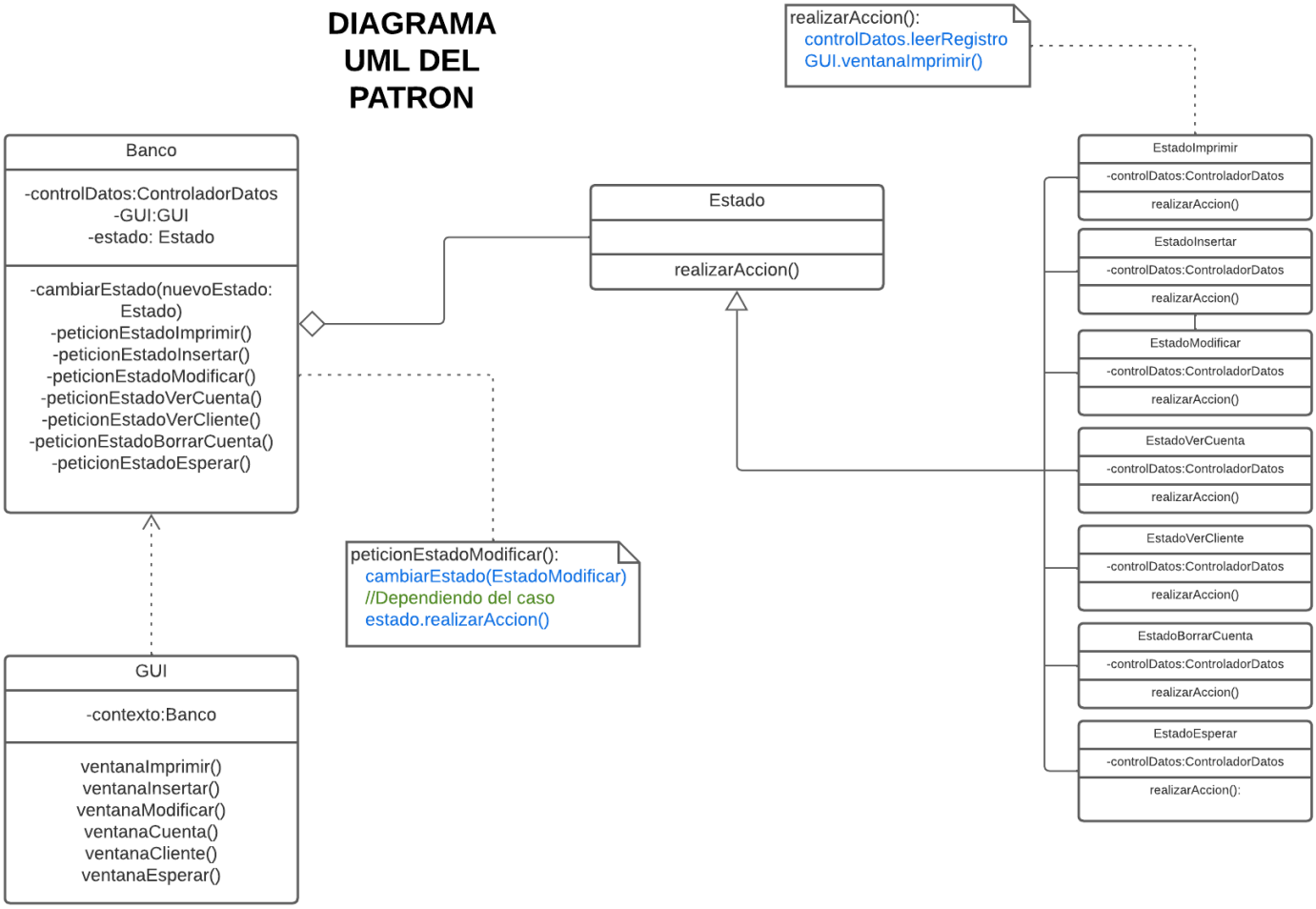
State: *es útil cuando los objetos están previstos cambien su comportamiento, en este caso la aplicación misma es la que cambiará constantemente su comportamiento dependiendo del estado interno en el que se encuentre, se utiliza en el proyecto porque que me permite desacoplar la lógica del comportamiento de los objetos y facilita la extensibilidad al darme la posibilidad de añadir nuevos estados y con ello nuevas funcionalidades; por ejemplo, si se pretende en un futuro que el programa tenga diferentes tipos de usuario, este patrón permite llevar un control de las funcionalidades accesibles para cada usuario.*

Como puntos adicionales a remarcar del patrón en la implementación del proyecto:

Simplifica enormemente la lógica condicional: cada estado se responsabiliza de su propio comportamiento

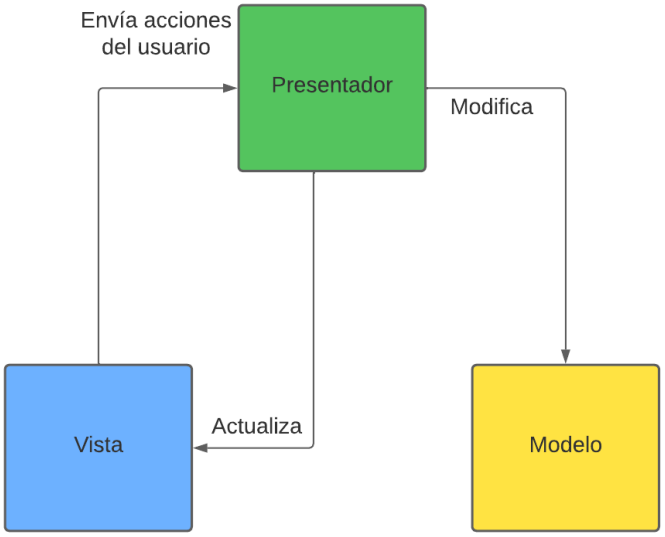
Interfaz de usuario dinámica: cada estado de la aplicación requiere diferentes opciones y campos, además de que con buenas prácticas de programación se puede reutilizar el código de forma que solo se habiliten y deshabiliten funcionalidades de la interfaz

Control de transiciones de estado: al usar este patrón me aseguro de que los cambios sean predecibles y de esa forma evito las situaciones de inconsistencia o de comportamiento inesperado



Modelo-Vista-Presentador: *se utiliza debido a la presencia de una interfaz de usuario (GUI), separa la lógica de presentación y la lógica del negocio. El modelo representa los datos y la lógica de negocio, la vista es responsable de mostrar la interfaz de usuario y el presentador actúa como intermediario entre el modelo y la vista, gestionando la comunicación y la actualización de datos.*

PATRON DE DISEÑO MVP
APLICADO AL PROYECTO



Presentador: Está presente en el módulo control del proyecto, se encarga de llevar a cabo las operaciones internas del programa.

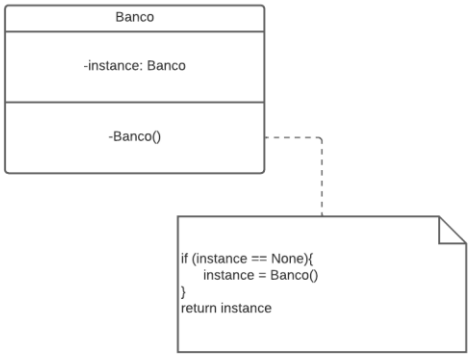
Modelo: Está presente en el módulo con el mismo nombre(Modelo) y se encarga de representar los agentes involucrados en la realidad, cuentas y clientes.

Vista: Está presente en el módulo Vista e informa al usuario de las peticiones que esté solicita, modificando la interfaz que el mismo ve de acuerdo a lo que el presentador necesite.

Singleton: *Se utiliza para que Banco solo tenga una instancia disponible y siempre que se llame a Banco se acceda al mismo objeto, la ventaja de implementarlo es que tengo la certeza de que no tengo a Banco duplicado al cambiar de estado, de igual forma la manera en la que lo he implementado permite que se pueda llamar directamente al constructor Banco() y este siempre devolverá la misma instancia en caso de existir, evitando así el uso del get_Instance().*

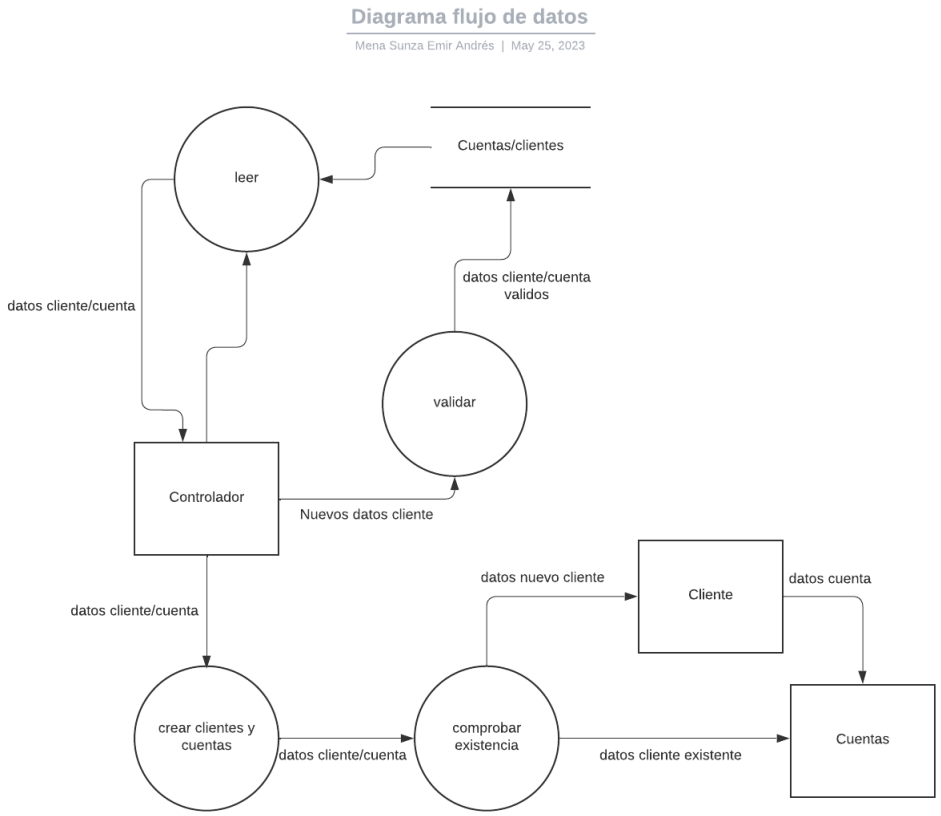
PATRON DE DISEÑO SINGLETON
APLICADO AL PROYECTO

```
class Banco:
    __instance = None
    def __new__(cls,*args, **kwargs):
        if (cls.__instance is None):
            cls.__instance = super(Banco,cls).__new__(cls)
        return cls.__instance
```



Implementación

Manejo de los datos: *Como toda aplicación, es necesario contar con una base de datos que se pueda actualizar y con ello reflejar los cambios, la base de datos del programa es un CSV, por lo que primero debe ser correctamente interpretado, ese trabajo lo lleva a cabo el controlador de datos que es llamado desde los estados del contexto (Banco)*



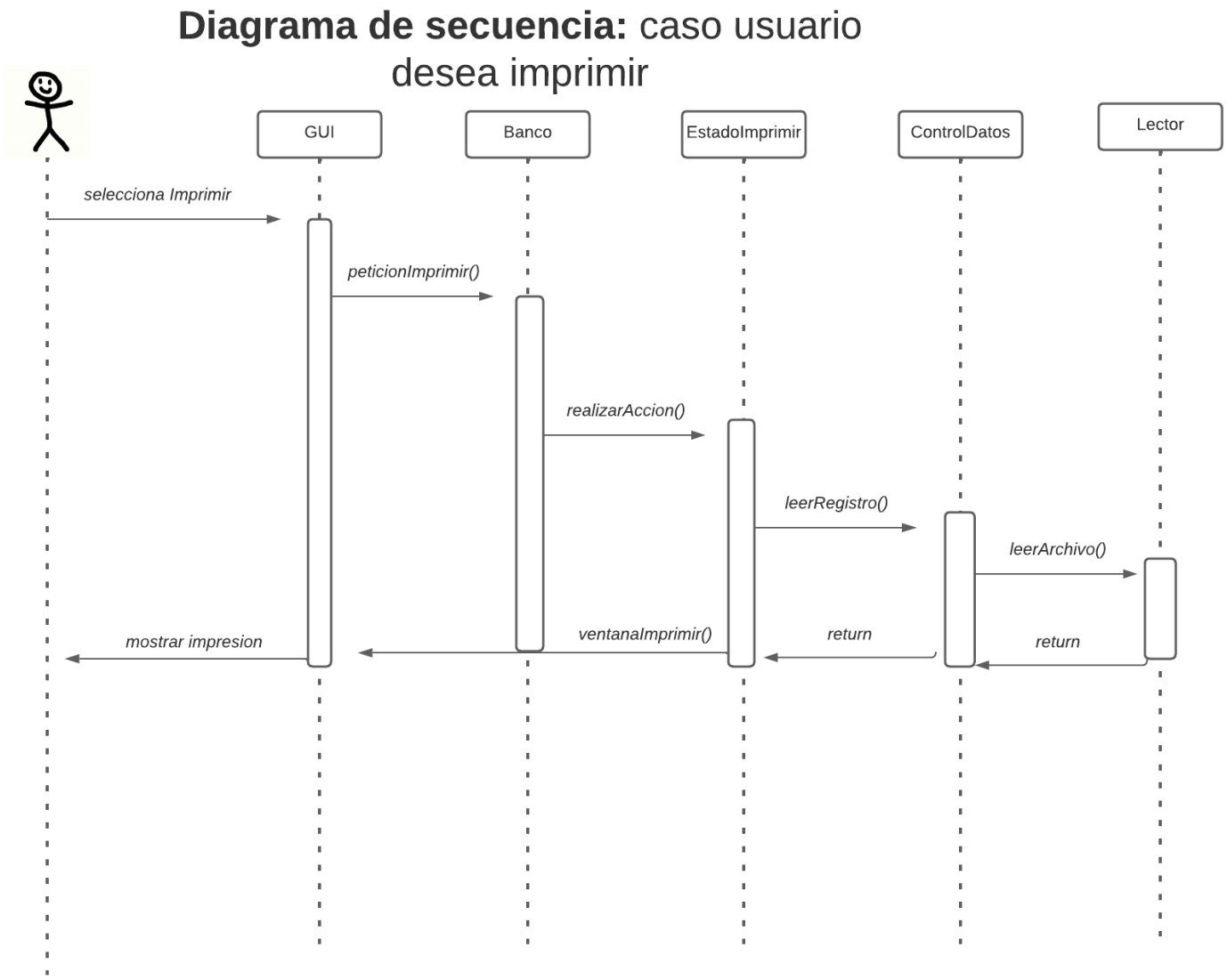
Descripción de las clases del programa:

Banco: *Esta es la clase principal del programa y que funge como contexto en el patrón de state, por si misma no lleva a cabo operaciones, sino que llama a los estados para que estos sean los encargados de ejecutar funciones que realicen las ya mencionadas operaciones.*

Estados*: Los estados del programa son el cerebro del programa debido a su relevancia en la lógica de este, ellos actualizan la interfaz gráfica y se encargan de llamar a controlador de datos para efectuar los cambios en la base de datos*

GUI: esta clase realiza cambios en la interfaz gráfica y se encarga de informar de cambios realizados por el usuario al modulo presentador, más precisamente al contexto.

Clientes y Cuentas: son el modelo, reflejan la interpretación de la base de datos y no tienen acceso a otras clases



Nota adicional: Soy consciente de que existen patrones que se adecúan más fácilmente al programa a implementar, sin embargo, opté por State para tratar de entenderlo mejor al tener que realizar modificaciones y acoplar tanto la lógica que ya tenía como el patrón en sí mismo, además que para añadir funcionalidades a las cuentas o tipos de clientes solo necesito transformar estas clases en contextos.