# Memoria del desarrollo de la aplicación.

**Oscar David Bocanegra Capera**

**Universidad Internacional de la Rioja**

**Javier Diaz Diaz**

**4 Septiembre 2023**

**Resumen.**

En este trabajo vamos a poder evidenciar como se le da solución a la situación problema planteada por medio de la programación orientada a objetos (POO) utilizando métodos como la herencia, la abstracción, las interfaces y de la misma manera desarrollando un diagrama UML en el cual vamos a poder darle una mejor claridad al ejercicio para así de la misma manera plasmarlo en el código, así que en este documento vamos a poder ver capturas de pantalla del ejercicio mismo para luego así darle su debida explicación, para una mayor entendimiento del código para el lector y para dejarlo plasmado en forma de documentación.

**Ejercicio problema**

En un sistema bancario, se tiene diferentes productos que se le ofrecen a los clientes para facilitar su gestión económica y financiera. En ese orden de ideas, se identifican los siguientes aspectos que hacen parte de dicho sistema.

Cliente, del cual se conoce el documento de identidad, nombre, correo electrónico, número de celular y la dirección que presenta para comunicaciones y extractos

Cuenta de ahorro, la cual es un producto para el cliente y de la cuenta de ahorro se conoce, además de la información del cliente, el número de cuenta, fecha de apertura, porcentaje de interés que se aplica al ahorro y se refleja en el saldo cada mes y saldo. Por tal motivo, se debe implementar el proceso de cálculo de intereses mensuales.

Cuenta Corriente, que es un producto para el cliente y de la cuenta corriente se conoce, además de la información del cliente, el número de la cuenta, fecha de apertura, saldo, porcentaje de interés que se aplica al saldo de la cuenta cada mes y valor permitido de sobregiro. Por tal motivo, se debe implementar el proceso de cálculo de intereses mensuales.

Certificado de ahorro a término fijo, CDT, que es otro producto para el cliente y del CDT se conoce, además de la información del cliente, el número del CDT, fecha de apertura, número de meses (plazo), valor (monto) y el interés mensual que se pagará por este ahorro. Al CDT se le aplica una sola vez el porcentaje de interés mensual y se hace al momento de finalizar el plazo del producto. Por tal motivo, se debe implementar el proceso de cálculo del valor a entregar al cliente, que corresponde al monto, más la suma de los intereses reconocidos en el plazo estipulado en el CDT.

Tarjeta de crédito, que es un producto para el cliente y de la tarjeta de crédito se conoce, además de la información del cliente, el número de la tarjeta, fecha de apertura, fecha de vencimiento, interés a cobrar por el uso, cupo y valor utilizado. Al final de cada mes se calculan los intereses,

aplicando el porcentaje al valor utilizado. Por tal motivo, se debe implementar el proceso de cálculo de intereses mensuales.

Para este ejercicio, con base en la situación problema planteada, se solicita realizar el programa en Java, utilizando POO, aplicando temas como herencia, polimorfismo y si es viable interfaces y clases abstractas.

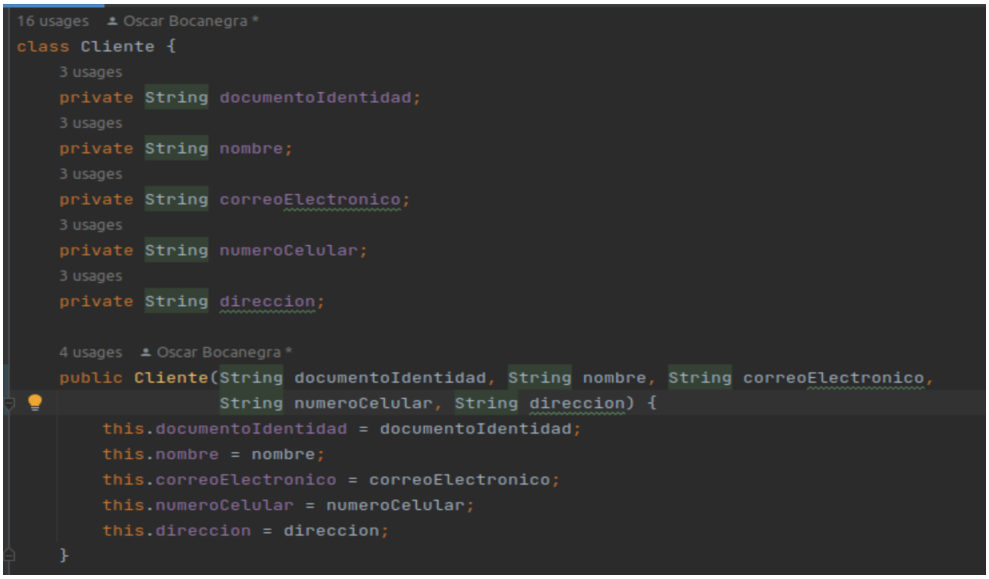
**Clases Principales**

**Cliente**

Como lo vimos en la lectura la clase ‘Cliente’ se va a basar en obtener la información más relevante del cliente como lo es el nombre, documento de identidad, correo electrónico, número de celular, y dirección, además de esto podemos aclarar que cada cliente puede tener una o varias cuentas bancarias asociadas.

Además esta la podemos tener como la principal ya que las demás clases van a necesitar de esta información.

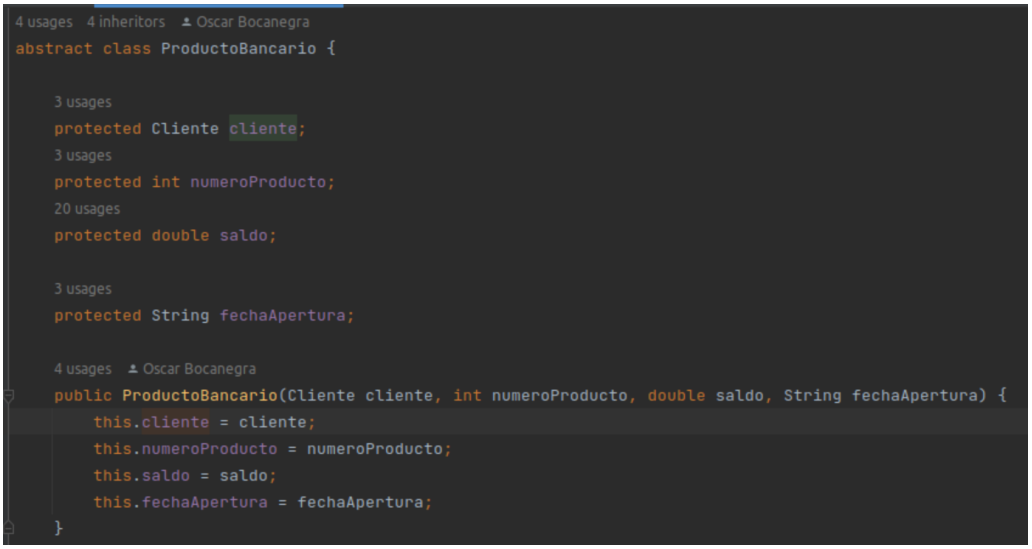
Y para resaltar en el código vamos a poder ver los getters y los setters los cuales van a ser utilizados ya que los atributos de la clase menciona los definimos como ‘private’

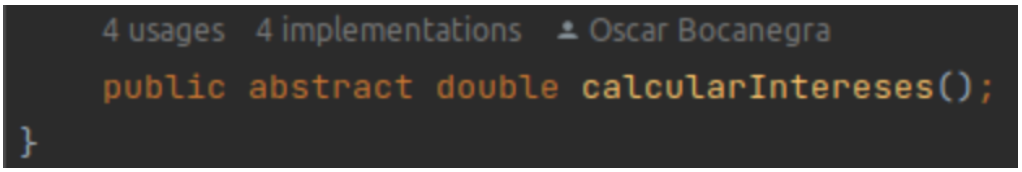


**Producto Bancario.**

Cuando vemos la clase ‘ProductoBancario’ podemos observar que es una clase abstracta ya que esta nos va ser útil para todos los productos bancarios que vamos a manejar en el presente ejercicio. Esta clase contiene los atributos compartidos como el cliente, el número del producto, el saldo y la fecha de apertura, además de esto también podemos ver que posee un método abstracto el cual es calcular interés con el fin de que este pueda ser implementado en las clases derivadas.

Y al igual que en las clases de este ejercicio posee sus getters y sus setters.



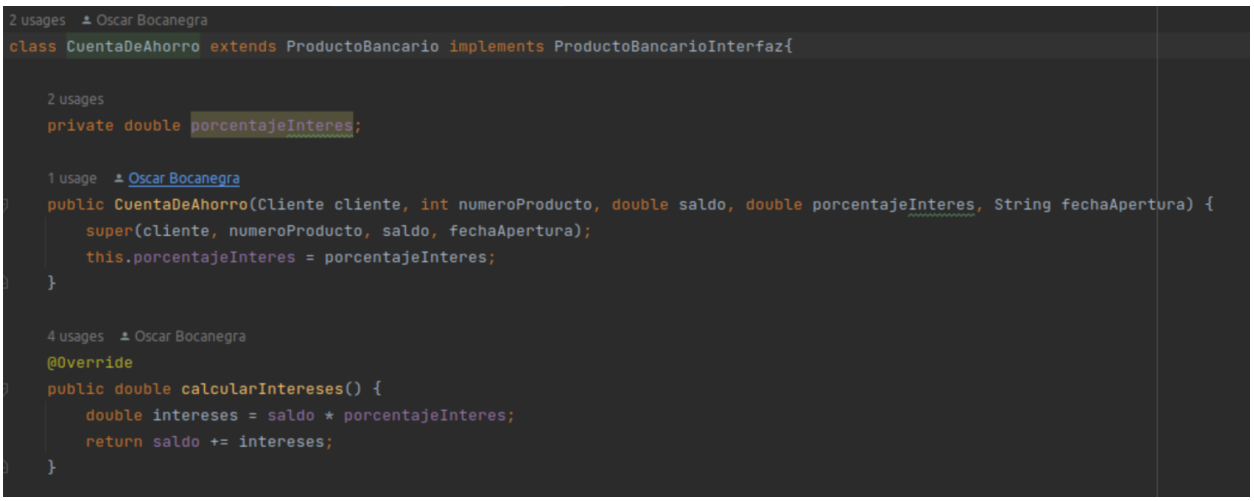



**Clases de productos bancarios**

**CuentaDeAhorros**

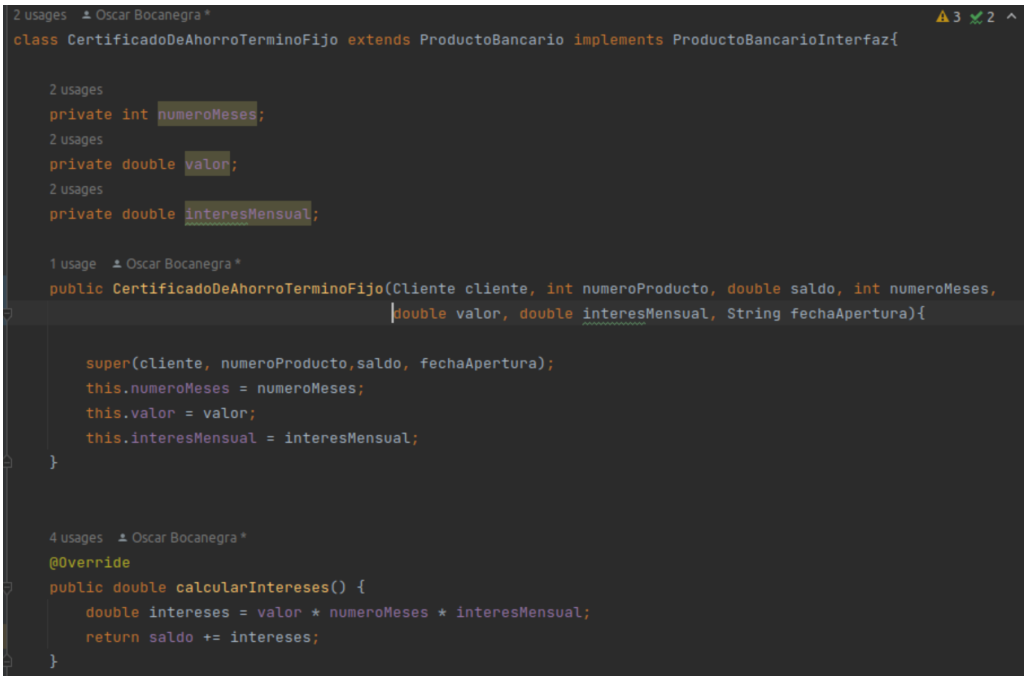
La clase CuentaDeAhorro representa una cuenta de ahorro y hereda de ProductoBancario. Almacena información específica de la cuenta de ahorro, como el porcentaje de interés mensual. La implementación del método calcularIntereses() calcula los intereses mensuales y los añade al saldo de la cuenta.

Y como vamos a poder ver en siguiente código esta clase tiene su metodo constructor, además de estos encontramos los @override el cual nos funciona para indicar que un método en una subclase está destinado a sobrescribir (override) un método con el mismo nombre y firma en su clase base o en una interfaz.




**CertificadoDeAhorroTerminoFijo**

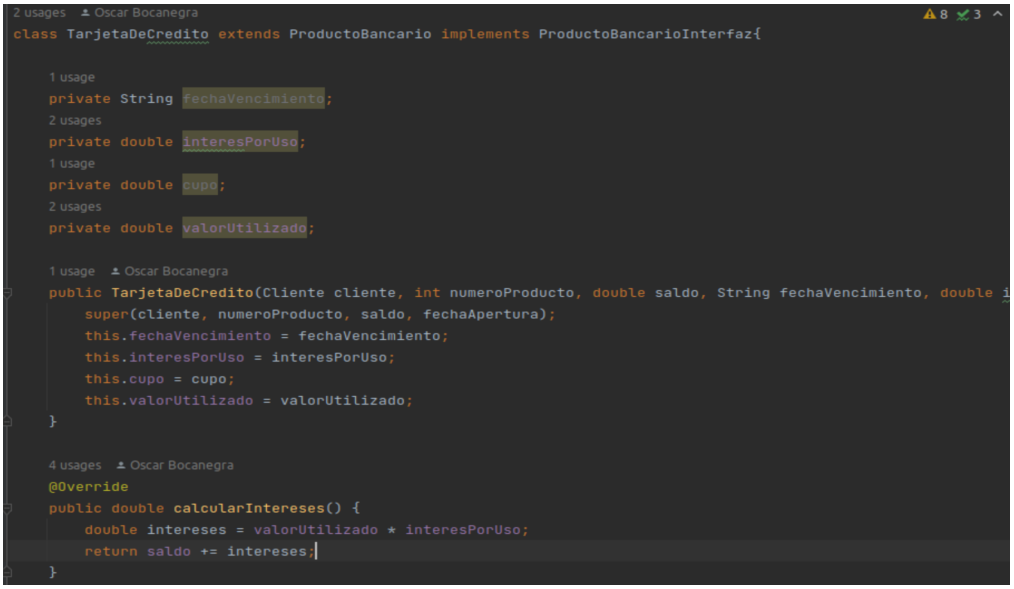
La clase CertificadoDeAhorroTerminoFijo representa un Certificado de Ahorro a Término Fijo (CDT) y hereda de ProductoBancario. Almacena información específica del CDT, como el número de meses (plazo), el valor (monto) y el interés mensual. El método calcularIntereses() calcula el valor total a entregar al cliente al finalizar el plazo, incluyendo los intereses.



```
2 usages  Oscar Bocanegra *                                                    ⚠ 3  ✓ 2
class CertificadoDeAhorroTerminoFijo extends ProductoBancario implements ProductoBancarioInterfaz{

    2 usages
    private int numeroMeses;
    2 usages
    private double valor;
    2 usages
    private double interesMensual;

    1 usage  Oscar Bocanegra *
    public CertificadoDeAhorroTerminoFijo(Cliente cliente, int numeroProducto, double saldo, int numeroMeses,
                                          double valor, double interesMensual, String fechaApertura){

        super(cliente, numeroProducto,saldo, fechaApertura);
        this.numeroMeses = numeroMeses;
        this.valor = valor;
        this.interesMensual = interesMensual;
    }


    4 usages  Oscar Bocanegra *
    @Override
    public double calcularIntereses() {
        double intereses = valor * numeroMeses * interesMensual;
        return saldo += intereses;
    }
```

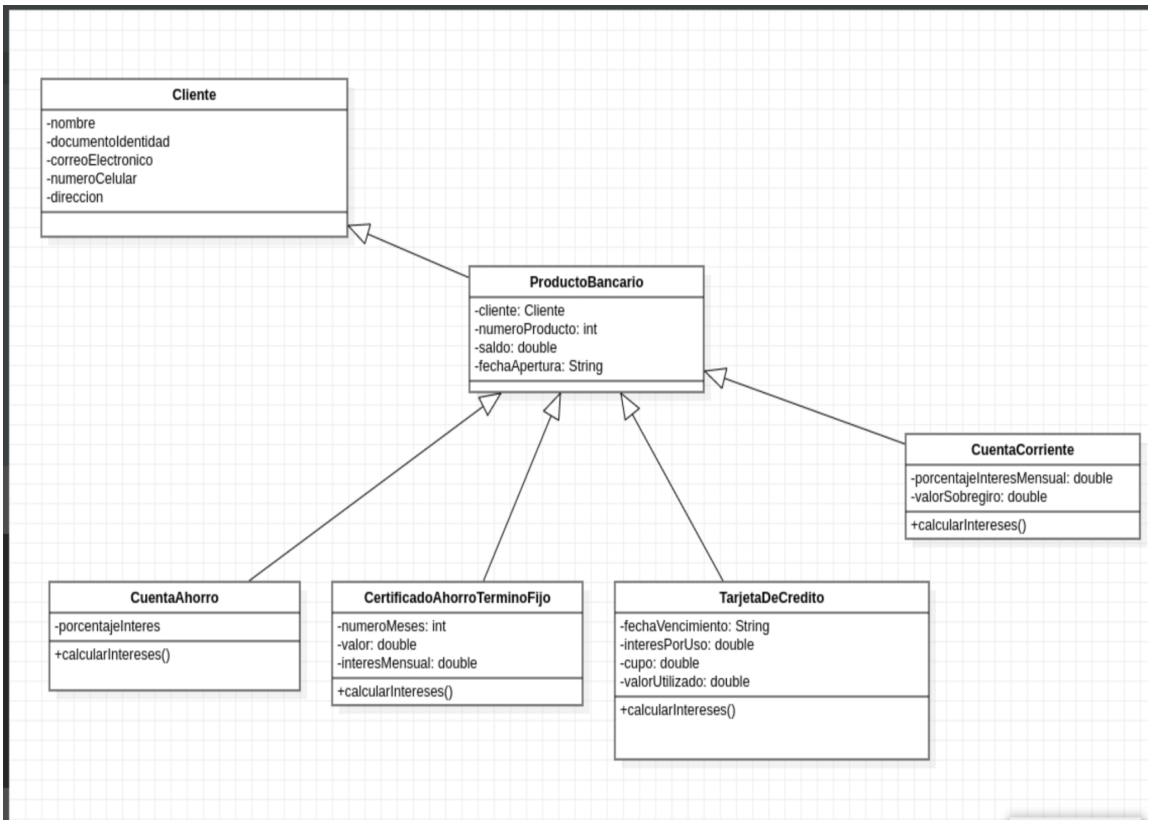
## Tarjeta de credito

La clase TarjetaDeCredito representa una tarjeta de crédito y hereda de ProductoBancario. Almacena información específica de la tarjeta, como la fecha de vencimiento, el interés por el uso, el cupo y el valor utilizado. El método calcularIntereses() calcula los intereses mensuales y los añade al saldo de la tarjeta de crédito.



```
2 usages  Oscar Bocanegra                                                      ⚠ 8  ✓ 3
class TarjetaDeCredito extends ProductoBancario implements ProductoBancarioInterfaz{

    1 usage
    private String fechaVencimiento;
    2 usages
    private double interesPorUso;
    1 usage
    private double cupo;
    2 usages
    private double valorUtilizado;

    1 usage  Oscar Bocanegra
    public TarjetaDeCredito(Cliente cliente, int numeroProducto, double saldo, String fechaVencimiento, double i
        super(cliente, numeroProducto, saldo, fechaApertura);
        this.fechaVencimiento = fechaVencimiento;
        this.interesPorUso = interesPorUso;
        this.cupo = cupo;
        this.valorUtilizado = valorUtilizado;
    }

    4 usages  Oscar Bocanegra
    @Override
    public double calcularIntereses() {
        double intereses = valorUtilizado * interesPorUso;
        return saldo += intereses;
    }
```

**Diagrama UML.**




En este trabajo también se puede destacar la importancia de un diagrama UML ya que antes de pasar al codigo deberiamos de plasmar y tener en claro que es lo que debemos de realizar y de qué manera para así de esta manera al momento de pasar al código va a ser mucho más fácil crear tanto las clases, los atributos y sus respectivos métodos.

**Conclusión.**

Con este trabajo podemos evidenciar como es el debido proceso para poder crear o modelar los objetos como se indica en la POO con sus respectivas clases y métodos ya sea por medio de los getters y setters o por medio de los override el cual nos ayudan a tener un código claro y de buenas prácticas a la hora de programar con Java y de programar acerca de la programación orientada a objetos