El diseño fue pensado para un sistema de software que contara con los siguientes requerimientos:

* Un CSV en un archivo/s de texto
* Un CRUD para el CSV
* Un reporte del CSV

El sistema está hecho para un banco, el cual cuenta con clientes y cuentas, con las siguientes “reglas de negocio”:

* Cada cliente tiene sus respectivos números de cliente único y sus números de cuenta único (los clientes también deben ser únicos)
* Cada cuenta tiene su respectivo saldo
* Cada cliente puede tener desde 0 o más cuentas

El diseño que decidí fue dividir en partes cada requerimiento y desarrollar, uno por uno, el plan y el proceso que utilizare para llevar a cabo su implementación, obviamente tomando en cuenta las reglas de negocio.

La Modularidad definida por el libro SWEBOK como dividir un programa informático grande en una serie de componentes más pequeños con nombre que tienen interfaces bien definidas que describen las interacciones de los componentes. La Modularidad es utilizada en el diseño porque se decidió dividir todo el sistema en clases más pequeñas, para así organizar y delegar las funcionalidades. En mi implementación se puede observar de forma explícita en:



La Abstracción definida por el libro SWEBOK como visión de un objeto que se centra en la información relevante para un fin determinado e ignora el resto de la información. La Abstracción es utilizada en el diseño porque las clases y por lo tanto los objetos del sistema tienen características específicas que lo distinguen de otro. En mi implementación se puede observar de forma explícita en:

```

//Objetos de las clases

        Report myReporte = new Report();

        CRUD mycrud = new CRUD();

        //Llamadas

        mycrud.create(123456789,"Octavio Villanueva");

        mycrud.create(123456789,"Octavio Villanueva");

        mycrud.create(987654321,"Gustavo Romero");

        mycrud.create(987654321,"Juan Xix");

        mycrud.create(456123987,"Fernando Valencia");

        mycrud.delete(456123987,"Fernando Valencia", "clients.txt");

        mycrud.update(987654321,"Gustavo Romero","Beatriz Poot");

        mycrud.create(123456789,645789123,2500);

        mycrud.create(987654321,645789123,4000);

        mycrud.create(987654321,147852963,4000);

        //mycrud.read("accounts.txt");

        //mycrud.read("clients.txt");

        //myReporte.printReport();

```

El Acoplamiento definido por el libro SWEBOK como medida de la interdependencia entre los módulos de un programa informático, consiste en identificarlas formas de conexión que existen entre módulos y la "fuerza" de estas conexiones. El Acoplamiento es utilizado en el diseño porque se hizo que todos los módulos o clases del sistema dependan uno del otro, de esta manera todo el sistema está conectado y cada clase necesita de otra. En mi implementación se puede observar de forma explícita en:

```

        String Clients ="clients.txt";

        Validations validate = new Validations();

        //Se verifica si el archivo existe para abrirlo, si no existe crea uno

        try (FileWriter fw = new FileWriter(Clients, true)){

            //Validación de existencia del cliente

            if(validate.existence(id,client,Clients)){

                StringBuilder sb = new StringBuilder();

                sb.append(id);

                sb.append(',');

                sb.append(client);

                sb.append(',');

                sb.append('\n');

                //Se inserta el nuevo cliente al final del txt

                fw.append(sb.toString());

                fw.close();

            }

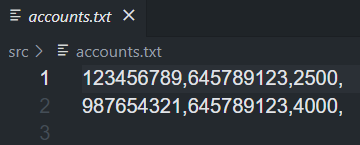
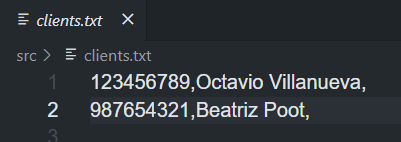
        }catch (Exception e) {

            System.out.println("Error: " + e.getMessage());

        }

```

Consideraría que realice una actividad de diseño llamada toma de decisiones y resolución de problemas, ya que tuve que decidir si seguir con mi diseño anterior o modificarlo generando el actual diseño y resolviendo los problemas que me causaba el diseño anterior. Esto debido a que, al principio para el CSV tenía planeado utilizar un solo archivo de texto que contuviera todos los datos separados por comas (número de cliente, nombre de cliente, número de cuenta, saldo) para así solo tener que trabajar con un CSV. Sin embargo, me di cuenta que era más práctico trabajar con dos archivos uno para los clientes y otro para las cuentas, de esta manera me evito muchas restricciones como estar repitiendo información cada vez que quiera agregar otra cuenta o que si borro una cuenta también se borre el cliente. Entonces, en el archivo para clientes solo tengo el número de cliente y el nombre del cliente; en el otro archivo para cuentas tengo el número de cliente (que indica a quien pertenece la cuenta), el número de cuenta y el saldo de dicha cuenta. En mi implementación se puede observar de forma explícita en:



Para el CRUD utilice:

* Dos métodos create: si no existe el archivo de texto lo crea y le agrega los datos dados o si existe solo le agrega los datos (para agregar datos se debe validar que no existan en el CSV), se usan dos métodos con el mismo nombre, pero con diferentes parámetros, no se puede usar el mismo debido a que las cuentas tienen un parámetro numérico más, el cual es el saldo.
* Un método read: lee el archivo de texto y lo imprime tal cual
* Un método update: modifica solo el nombre del cliente
* Un método delete: borra clientes o cuentas (las cuentas deben tener saldo de $0)

Para realizar el reporte fue sencillo solo se lee el archivo de clientes línea por línea y cada que se lee una línea del archivo de clientes es lee una todas las líneas del archivo de cuentas para que de esta manera se pueda saber si la cuenta corresponde al cliente, si es así se muestra en el reporte. El reporte consta de una lista de clientes y de sus respectivas cuentas, se muestra un cliente todas las cuentas que tiene y luego se pasa al otro cliente.

Una restricción de diseño definida en la presentación unidad 1 del curso de Diseño de Software como una limitación que detiene el espacio de la solución del problema de diseño, el cual es un elemento de diseño que así mismo se convierte en un error de diseño inferior definida en la presentación unidad 1 del curso de Diseño de Software como un error que no aborda de manera los requisitos de calidad especificados, ya que se vuelve ineficiente e inflexible, es el tener que sobrescribir los archivos de texto para poder implementar los métodos update and delete. Es una restricción puesto que el lenguaje de programación utilizado es java y java para poder modificar un archivo de texto debe copiar la información en otro, modificarla, borrar el archivo anterior y luego cambiar el nombre del nuevo archivo al que tenía el archivo anterior. Al menos esa fue la manera que utilice en mi implementación debido a que no encontré otra que me permitiera no sobrescribir. Ahora, se vuelve un error de diseño ya que si quisiera cambiar mi implementación para una base de datos no podría debido a la sobreescritura, deja de ser eficiente al toparse con esta barrera y flexible al no permitir esta modificación en un futuro. En mi implementación se puede observar de forma explícita en:

```

 //Se crea una variable File que contiene el nombre del archivo original

            File inFile = new File(file);

            //Se crea una variable File que contiene el nombre del archivo nuevo

            File tempFile = new File(inFile.getAbsolutePath() + ".tmp");

            BufferedReader br = new BufferedReader(new FileReader(inFile));

            PrintWriter pw = new PrintWriter(new FileWriter(tempFile));

            String line = null;

            String stringID=String.valueOf(id);

            //Se lee el archivo original para escribir en el nuevo

            //Si se lee los mismos datos dados en el archivo original entonces no se escriben en el nuevo

            //De esta manera parecera que se borro

            while ((line = br.readLine()) != null) {

                if(file.equals("clients.txt")){

                    if (!line.trim().equals(stringID+","+data+",")) {

                        pw.println(line);

                        pw.flush();

                    }

                }

                if(file.equals("accounts.txt")){

                    if (!line.trim().equals(stringID+","+data+",0,")) {

                        pw.println(line);

                        pw.flush();

                    }

                }

            }

            pw.close();

            br.close();

            //Borrar el archivo origional

            inFile.delete();

            //Renombrar el archivo nuevo con el nombre del archivo original

            tempFile.renameTo(inFile);

```

Cambios requeridos en la implementación:

* Conexión a una base de datos (BD)
* Mecanismo de seguridad vía encriptación para la base de datos
* Generar reporte en un PDF

Para conectar el sistema a una BD se necesita primero un servidor y segundo cambiar el diseño de los métodos delete y update, para evitar la sobreescritura.

Para el mecanismo de seguridad se necesita definir 3 algoritmos de encriptación que se utilizarán para hacer la conexión a la BD. Como extra, se puede observar en el código fuente, dos archivos (Encrypt y Decrypt) no conectados al sistema que se utilizara como base para la implementación del mecanismo de seguridad y la conexión del sistema a la BD.

Para generar el reporte del banco en PDF según mi investigación existen 2 opciones:

* Utilizar una API: esta API se encarga de crear el PDF y solo hay que llamarla
* Utilizar una librería: la librería se debe instalar de forma local y se debe definir en la clase que se implementara para poder utilizarla

Es cuestión de decidirme por una de las dos opciones, considerarla en el nuevo diseño o al modificar el actual.