PROYECTO FIS

- Participantes:
    - ·WueLiang Chen Bao
    - ·Jorge González Delgado
- Índice:

- Links:

-Enlace al proyecto RedMine:
https://fis.etsisi.upm.es/projects/citim22-09-upmfit?jump=welcome

-Enlace al proyecto GitLab: https://gitlab.etsisi.upm.es/grupo-fis/practica-fis

1. Valoración crítica:

Valoración crítica sobre la aportación que ha tenido para la implementación la realización de las fases previas de ingeniería de software.

- Fase 1 : . Especificación de Requisitos

En esta fase lo que hemos hecho es extraer los requisitos a partir del enunciado proporcionado (UPMFit), estos requisitos son los que definieron lo que tiene que hacer el software en su fase final. Buscando satisfacer las necesidades del cliente , en este caso la UPM.
El primer paso consiste en recopilar toda la información relevante sobre el software que se va a desarrollar, a través de un análisis del enunciado . Posteriormente, se lleva a cabo un proceso de descomposición y categorización de los requisitos, con el fin de identificar las funcionalidades y características esenciales que el software debe poseer ,introduciendolas dentro del redmine siguiendo el estándar IEEE 830.

Por último se ha hecho un Mockup del proyecto para mostrar una ejecución de lo que sería el problema y tener una idea general de como funciona.

- Fase 2 : Análisis

Durante la fase de análisis, se lleva a cabo un análisis exhaustivo de las necesidades del cliente y se evalúan los requisitos propuestos en la fase de especificación. El objetivo principal es comprender a fondo el problema que se busca resolver con el software y determinar la mejor manera de abordarlo.
En esta fase, se realiza una búsqueda de un modelo que represente el problema real, en este caso, el diseño de un gimnasio. Una de las herramientas utilizadas es el diagrama de clases de análisis, que muestra las clases del sistema, sus atributos y las relaciones estructurales y de herencia entre ellas. Este diagrama tiene como objetivo principal capturar y comunicar los conceptos clave y las relaciones en el dominio del problema.
Además del diagrama de clases, se utilizan diagramas de casos de uso para representar las funcionalidades específicas y comprender cómo funcionan los requisitos y características del software. Estos diagramas proporcionan una representación visual de cómo interactúa el usuario con el sistema y cómo se cumplen los requisitos en diferentes escenarios.
Asimismo, se elaboran diagramas de casos de uso extendidos, que representan la ejecución de características específicas en detalle. Estos diagramas permiten comprender cómo se desarrollan los casos de uso en situaciones particulares, identificando los pasos y acciones necesarios para lograr los resultados deseados.

- Fase 3 : Diseño

En esta fase lo que se busca es centrarse en cómo funciona la aplicación por dentro.
Se hace un refinamiento del diagrama de clases de análisis para convertirlo en un diagrama de clases de diseño , esto se logra añadiendo los tipos de variable a los atributos de cada clase , introduciendo las funciones necesarias para cada clase , incorporación de interfaces ,introducción de controladores y de clases de sistema.
También se hace uso de patrones de diseño y patrones arquitectónicos , en nuestro caso hemos seguido el patrón SOLID y el patrón Model View Controller (MVC).
También se han creado un diagrama de componentes y un diagrama de despliegue

En resumen , en la fase de especificación extrajimos los requisitos que debía de tener el software y los introdujimos en el Redmine para poder documentarlo , la fase de análisis nos dió una estructura básica sobre cómo sería el software a través del diagrama de clases de análisis y nos dio una representación visual de cómo funcionan distintas partes del programa a través de los diagramas de caso de uso y los diagramas de caso de uso extendido y finalmente en la fase de diseño refinamos el diagrama de clases de análisis a uno de diseño para entender cómo funciona la

aplicación por dentro y por lo tanto codificarla después en la fase de implementación.

2. Pruebas de caja negra de la clase Cliente:

```
public void setEdad(int edad){
```

ENTRADA	CLASES DE EQUIVALENCIA VÁLIDAS	CLASES DE EQUIVALENCIA INVÁLIDAS
Edad	V1: 14	N1: -9

CP1: V1 CP2: N1

```
public void setPeso(int peso){
```

ENTRADA	CLASES DE EQUIVALENCIA VÁLIDAS	CLASES DE EQUIVALENCIA INVÁLIDAS
Peso	V1: 62	N1: -7

CP1: V1 CP2: N1

```
public void setSexo(String sexo){
```

ENTRADA	CLASES DE EQUIVALENCIA VÁLIDAS	CLASES DE EQUIVALENCIA INVÁLIDAS
Sexo	V1: Hombre	N1:

CP1: V1 CP2: N1

```
public void setTarjeta(int tarjeta){
```

ENTRADA	CLASES DE EQUIVALENCIA VÁLIDAS	CLASES DE EQUIVALENCIA INVÁLIDAS
Tarjeta	V1: 405	N1: -54

CP1: V1 CP2: N1

```
public void setId (int id) {
```

ENTRADA	CLASES DE EQUIVALENCIA VÁLIDAS	CLASES DE EQUIVALENCIA INVÁLIDAS
id	V1: 405	N1: -8

CP1: V1 CP2: N1

```
public void setDni(String dni){
```

ENTRADA	CLASES DE EQUIVALENCIA VÁLIDAS	CLASES DE EQUIVALENCIA INVÁLIDAS
DNI	V1: 28302716D	N1:

CP1: V1 CP2: N1

```
public void setNombreUsuario(String nombreUsusario){
```

ENTRADA	CLASES DE EQUIVALENCIA VÁLIDAS	CLASES DE EQUIVALENCIA INVÁLIDAS
Nombre	V1: PepeUPM	N1:

CP1: V1 CP2: N1

```
public void setContrasena(String contrasena){
```

ENTRADA	CLASES DE EQUIVALENCIA VÁLIDAS	CLASES DE EQUIVALENCIA INVÁLIDAS
Contraseña	V1: password	N1:

CP1: V1 CP2: N1

```
public void setCorreo(String correo){
```

ENTRADA	CLASES DE EQUIVALENCIA VÁLIDAS	CLASES DE EQUIVALENCIA INVÁLIDAS
Correo	V1: pepe@upm.es	N1:

CP1: V1 CP2: N1

```
public void setNombre(String nombre){
```

ENTRADA	CLASES DE EQUIVALENCIA VÁLIDAS	CLASES DE EQUIVALENCIA INVÁLIDAS
Nombre	V1: Pepe	N1:

CP1: V1 CP2: N1

3.Trazabilidad:

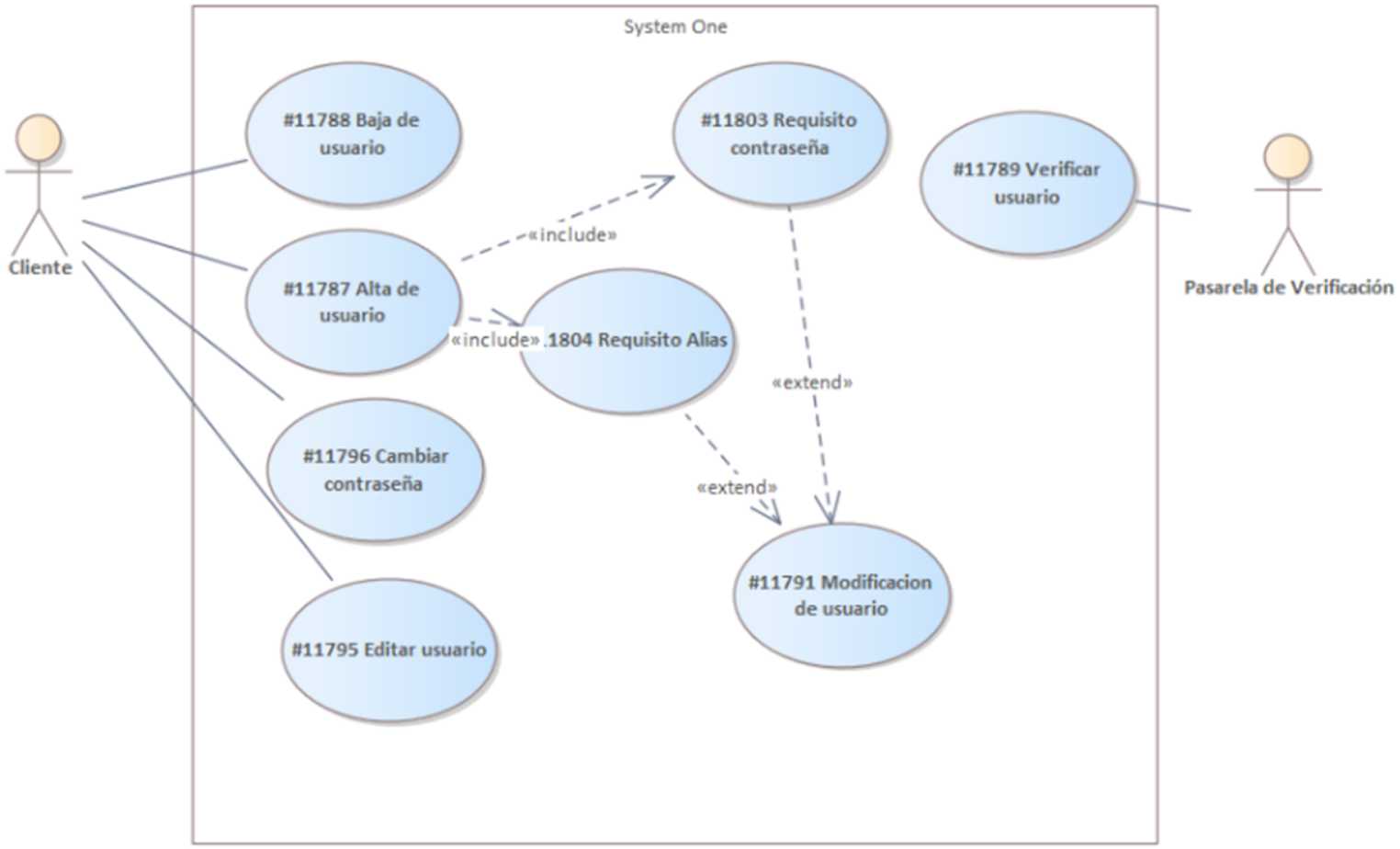
-Requisito funcional:

Descripción Citar

Esta función permitirá añadir un nuevo usuario al sistema de Gestión de Usuarios. Para poder realizar el alta de un usuario se deberán realizar las siguientes funciones:

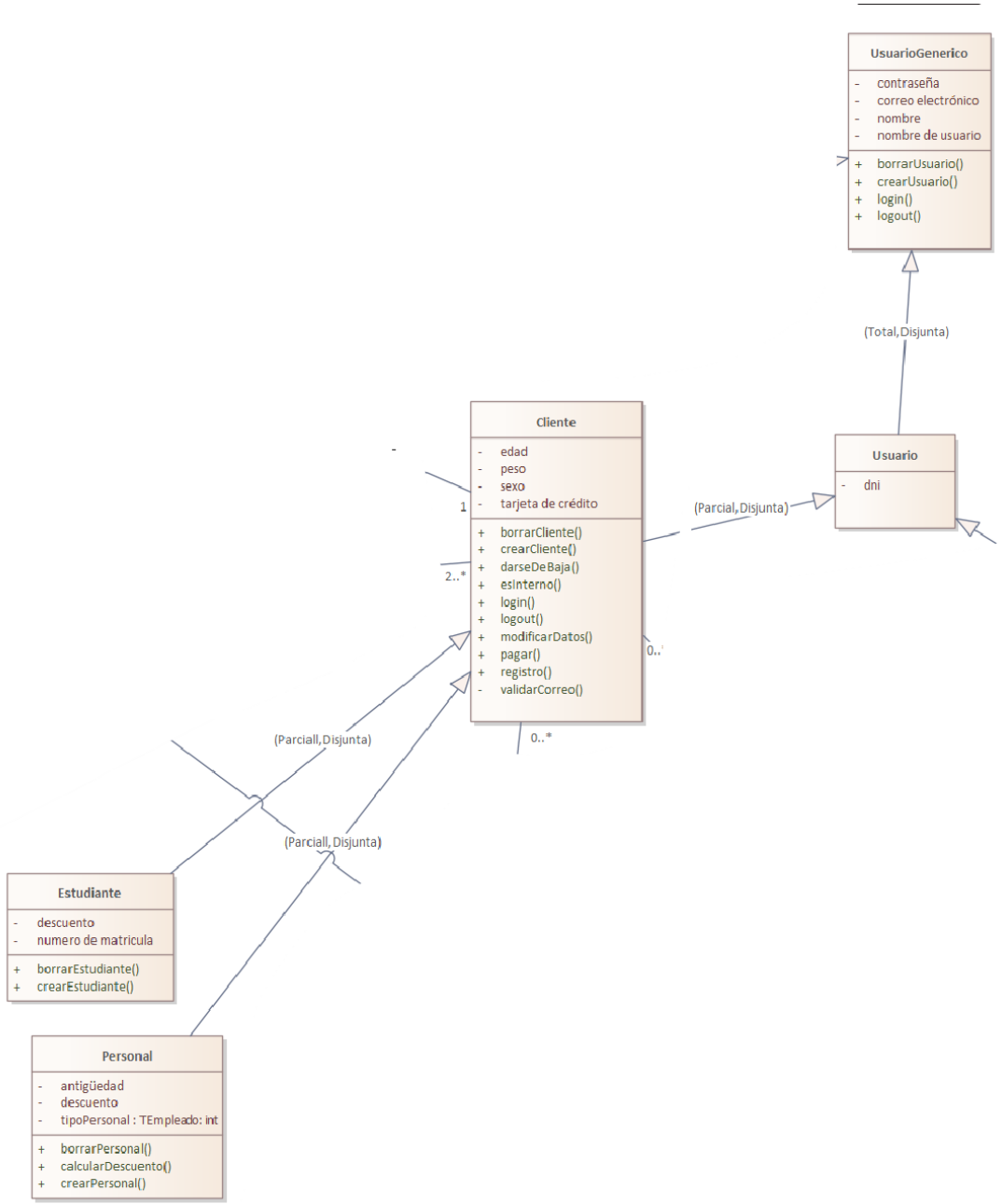
1. Pedir los datos necesarios para los atributos de cada usuario: alias, nombre completo, clave, correo universitario.
2. Comprobar que los datos cumplen las restricciones de cada uno de los atributos de un usuario.
3. Comprobar que el alias y el correo universitario no está repetido
4. Almacena los datos

-Casos de Uso: (En esta fase del proyecto denominamos a ‘Alta Cliente’ como ‘Alta Usuario’)



-Análisis:

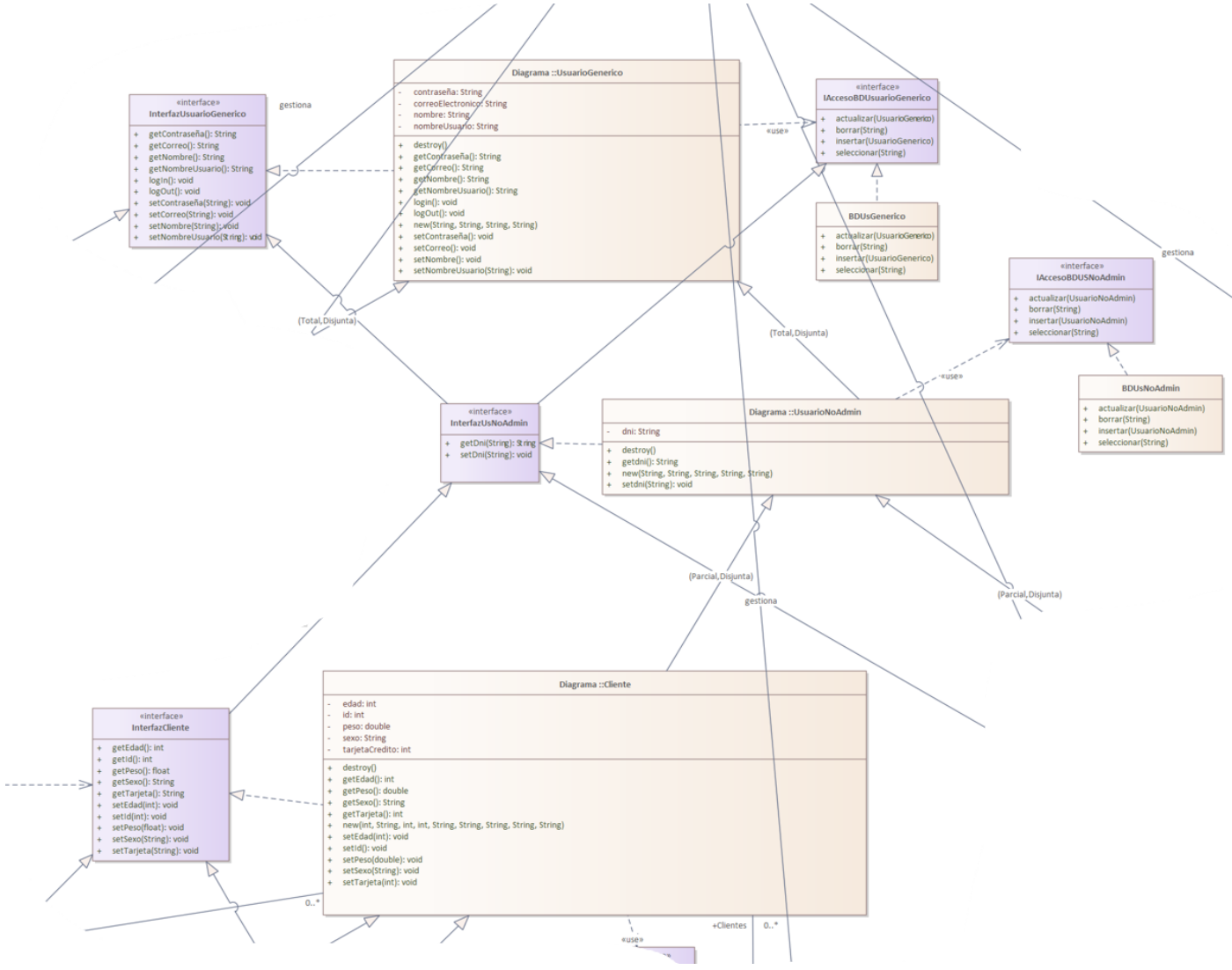
Después de sacar los requisitos , conseguimos las siguientes clases de análisis a partir del enunciado , se dice que los clientes pueden ser externos o internos por lo que se hace una herencia de Cliente a Estudiante y a Personal

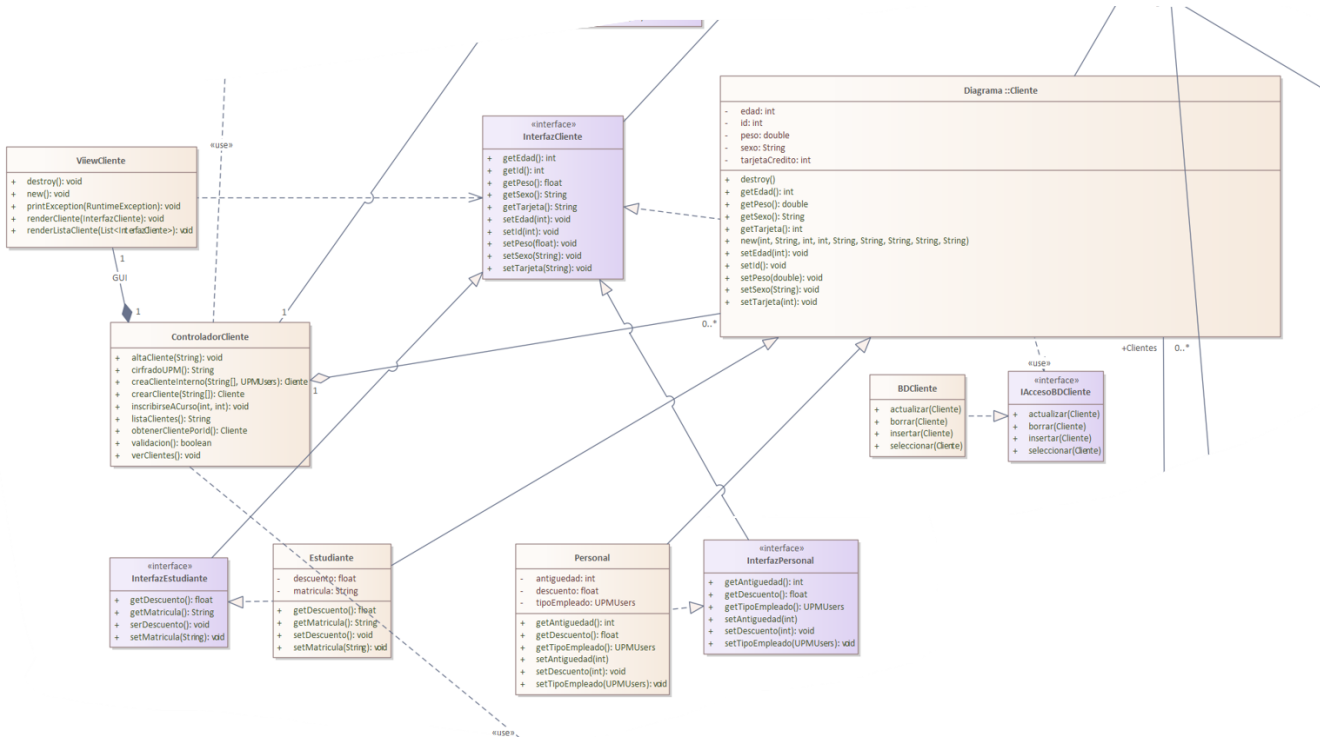


-Diagrama de diseño:

Se hace un refinamiento de análisis a diseño incluyendo los tipos de datos en las variables , se añaden interfaces y se aplican patrones de diseño.

El Model View Controller es incluido para las clases que lo necesitan en nuestra aplicación





-Implementación:

La operación de dar de alta un cliente nuevo se hace desde la clase ControlladorCliente

```
public class ControladorCliente {


 public void altaCliente(String datos) {
     try {
     String info[] = datos.split(",");

     if (validacion(info[6])) {// comprueba si es de la UPM
     UPMUsers rol = getRol(info[6]);
     Cliente c = creaClienteInterno(info,rol);

     c.setId(clientes.size());
     clientes.add(c);


     }

     else {
         Cliente c = creaCliente(info);
         c.setId(clientes.size());
         clientes.add(c);
     }
     }catch (RuntimeException e) {
         ViewCliente.printException(e);
     }

 }
```

Se crea un Cliente o un ClienteInterno dependiendo de si se recibe un correo electrónico de la UPM o uno cualquiera

```
private Cliente creaCliente(String[] info) {

    //la contraseña se cifra dentro del modelo
    return  new Cliente (Integer.parseInt(info[0]),info[1],Integer.parseInt(info[2]),
            Integer.parseInt(info[3]),info[4],info[5],info[6],info[7],info[8]);
}


private Cliente creaClienteInterno(String[] info,UPMUsers rol) {
    if (rol == UPMUsers.ALUMNO) {
        Estudiante c = new Estudiante(Integer.parseInt(info[0]),info[1],Integer.parseInt(info[2]),Integer.parseInt(info[3]),
                info[4],info[5],info[6],info[7],info[8],info[9]);
        return c;
    }
    else {
        Personal c = new Personal(Integer.parseInt(info[0]),info[1],Integer.parseInt(info[2]),Integer.parseInt(info[3]),
                info[4],info[5],info[6],info[7],info[8],Integer.parseInt(info[9]));
        return c;
    }

}
```

Se necesita la función validacion para validar si el correo proporcionado es de la UPM , si no fuese el caso se crearia un cliente externo , en caso contrario se crea un Alumno o un Personal dependiendo del rol asignado

```
private UPMUsers getRol(String correo) {
    // TODO Auto-generated method stub
    UPMUsers rol = ObtencionDeRol.get_UPM_AccountRol(correo);

    return rol;
}
private boolean validacion (String correo) {
    Autenticacion autentica = new Autenticacion();
    return autentica.existeCuentaUPM(correo);
    }
```

-Pruebas: