Especificación de Diseño de Software

MiPandilla

Versión <1.0> <Diseño>

Alejandro Bermejo Vialas, Mariadalit Rosales Rodríguez, Mariano González Salazar, Jorge Peláez González, Saad Ziyad Khalid.

Universidad Carlos III de Madrid

12/11/2015

Contenido

1. Introducción

Este Documento de Diseño de Software (SDD) está adaptado para cumplir con las recomendaciones del IEEE1016 [3]. Es recomendable leer previamente los documentos SPMP [1] y ERS [2] para la completa comprensión del sistema. En este documento se da por hecho que el lector conoce de antemano el contexto, requisitos, casos de uso, etc; descritos en dichos documentos.

En primer lugar, se realiza una completa y detallada descripción del funcionamiento esperado y estructura del Software a diseñar. Se utilizarán diagramas para hacer más sencillo el entendimiento de estos procesos.

También, se describe un Plan de Pruebas minuciosamente diseñado para comprobar en su totalidad el comportamiento del programa, de tal manera que sea lo más robusto y eficiente posible.

Además se ha planificado una metodología de trabajo de forma que el equipo sea lo más eficiente posible, quedando un resultado claro en el código, y sencillo para el usuario final en un tiempo reducido.

Al final del documento se encuentra un glosario de términos para facilitar la lectura de este documento.

2. Descripción del sistema

- Reutilización de software: El software proporcionado por TeLEDmíticos, se entrega tal cual, corriendo con la actualización y mantenimiento la propia EMPRESA CLIENTE. Aun así, el código se entregará de tal manera que la aplicación de actualizaciones se haga de forma sencilla y sin modificar el código original en gran medida.
- Planes futuros para extender o mejorar el software: Siempre que se disponga capacidad de almacenamiento y procesamiento suficiente, así como una infraestructura que provea de conectividad vía Internet, será posible la expansión de MiPandilla a un público más amplio. Para realizar este propósito, no será necesario ningún cambio, aunque sí será necesario prever el número de usuarios, y expandir el almacenamiento en caso necesario. Una excesiva carga sobre un almacenamiento reducido puede causar problemas de seguridad de los que TeLEDmíticos no se hace responsable.
- Paradigmas de la interfaz de usuario: La interfaz de usuario será pseudográfica, es decir: se basará en una consola de comandos en la que el usuario introduce órdenes, pero observará una interfaz cómoda basada en diseño ASCII. El usuario irá navegando por un sistema de menús en los que debe introducir dónde quiere ir mediante su dispositivo de entrada de texto.

- Detección de errores: La detección de errores se hará en una primera instancia con la plataforma Eclipse (errores sintácticos). Tras ello, parte del equipo irá depurando cada funcionalidad de la aplicación, de tal manera que el programa considere todas las excepciones posibles. Además, si algún usuario detecta una mala funcionalidad del código dentro del primer mes de comercialización, podrá ponerse en contacto con bugs@teledmiticos.moe y se procederá a su subsanación.
- Utilización de ficheros de dato: La aplicación MiPandilla está implementada sobre un sistema de ficheros de datos sostenido por EMPRESA CLIENTE, en la que se almacenarán todos los datos de usuarios y eventos. EMPRESA CLIENTE deberá hacerse cargo de la privacidad de sus usuarios. Los datos serán almacenados sin ningún tipo de cifrado, legible sin necesidad de la apertura del programa; con excepción de la contraseña, que será guardada con encriptación SHA-512.

3. Consideraciones de diseño

En este apartado, describimos los aspectos generales en cuanto al diseño y desarrollo de nuestra aplicación. Se puede observar que escogimos el patrón MVC, dado que nos parecía el mejor para llevar a cabo la implementación de la aplicación.

Dicha implementación estará dividida en clases, que contendrán los métodos que aparecen definidos en las respectivas interfaces. Cada método y cada porción de código que resulte importante en el funcionamiento de la aplicación, irá debidamente comentado para hacer que el código sea más legible y comprensible.

3.1. Supuestos y dependencias

El sistema operativo para el que realizaremos la aplicación del proyecto será un entorno de Linux.

Programaremos la aplicación en lenguaje ***Java*** y utilizaremos como entorno de desarrollo la herramienta Eclipse.

Usaremos ***ficheros de texto*** como repositorio para la aplicación, que nos servirán para almacenar la información necesaria para el correcto funcionamiento de la misma (Información de usuarios/eventos/comentarios).

3.2. Restricciones generales

Las restricciones que tendremos en cuenta son las siguientes:

- **S.O:** La aplicación correrá sobre un sistema operativo Linux. Con esto, aseguramos el correcto funcionamiento de determinados comandos o llamadas al sistema que pueda realizar nuestra aplicación.

- **Interfaz:** el usuario se comunicará con la aplicación a través de *terminal* mediante teclado. Dicha interfaz será lo suficientemente clara como para que los usuarios no tengan ningún problema a la hora de interactuar con la aplicación.

- **Persistencia:** La aplicación empleara *ficheros de texto* para almacenar todos los datos necesarios a modo de DDBB. Esto supone, que el ordenador donde se ejecute el programa deberá contener siempre los ficheros generados para el correcto funcionamiento de la aplicación.

- **Hardware:** la aplicación no tendrá unos requerimientos de sistema demasiado exigentes, en lo que a *memoria ram* o *velocidad de procesador* se refiere, y podrá ser utilizada en cualquier ordenador sobremesa o portátil que tenga instalado el sistema operativo que indicamos en el primer apartado.

3.3. Metodología de desarrollo

El *Gestor de Desarrollo*, en base a lo establecido en el documento ERS, asignará las tareas a realizar por cada componente del grupo para llevar a cabo la implementación de la aplicación. Teniendo en cuenta que el grupo de trabajo está formado por cinco personas, la asignación de tareas será la siguiente:

- 1 persona se encargará de la programación de las clases Main, View y Controller.
- 1 persona se encargará de la programación de las clases de los objetos y sus respectivas interfaces.
- 1 persona se encargará de probar que la funcionalidad de las clases implementadas es la correcta.
- 1persona se encargará de la búsqueda de posibles fallos y mejoras del código.
- 1 persona se encargará de la correcta documentación (comentarios) del código.

El patrón de diseño que seguirá nuestro proyecto es el MVC (Model–View-Controller) que se puede observar en la *Figura 1*. Este patrón nos permitirá separar los datos y la lógica de la aplicación, de la interfaz de usuario.

Patrón MVC

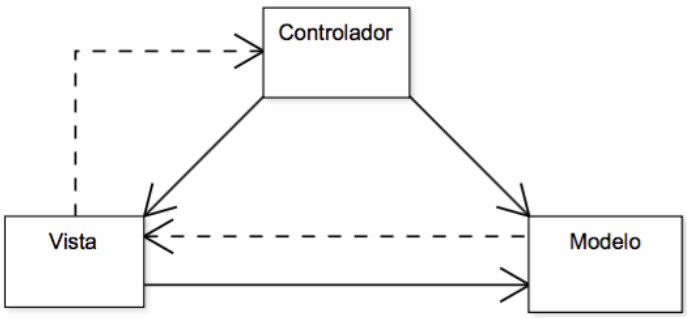
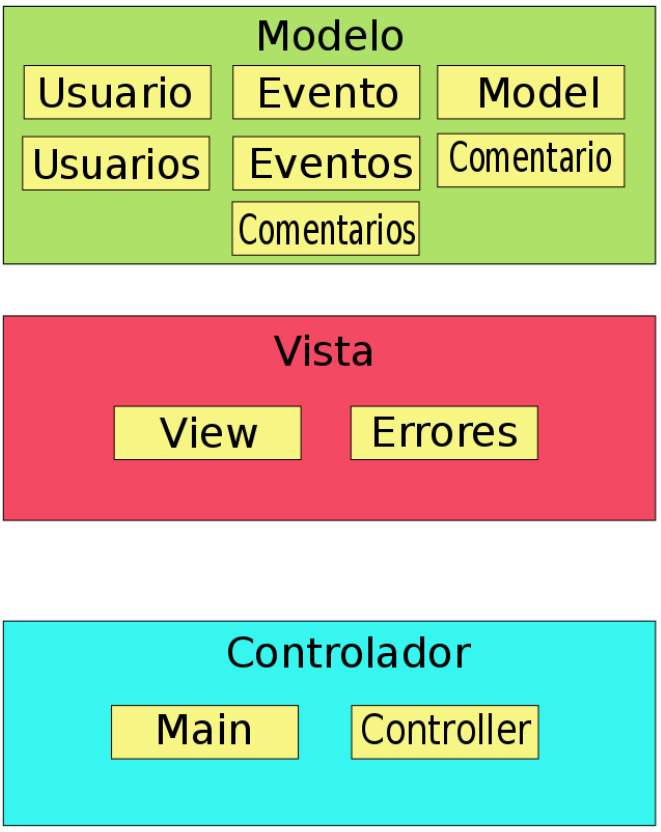


Figura 1.Estructura del patrón MVC

4. Arquitectura del sistema

De acuerdo al modelo MVC, la agrupación de clases de nuestro sistema es la siguiente:



Modelo

En el Modelo encontramos toda la lógica sobre la que gira nuestra aplicación, es decir, en él encontramos la información con la que nuestro sistema opera y la gestión sobre la misma. Cabe destacar que tanto las peticiones de acceso como las peticiones de manipulación de dicha información llegan al Modelo a través del Controlador. En este grupo contamos con las siguientes clases:

- Clase Model: en ella encontramos todos los métodos necesarios para gestionar las acciones que el usuario haya realizado a través de la terminal, como por ejemplo los métodos comentarPerfil() o eliminarCuenta().
- Clase Usuario: en ella se encuentra toda la información relacionada con cada usuario, como es el nombre identificativo, la contraseña, lugar de residencia o estado civil.
- Clase Evento: en ella se encuentra toda la información relacionada con cada uno de los eventos que se encuentran en nuestra aplicación, como puede ser la fecha de dicho evento o el lugar de realización del mismo.

- Clase Comentario: contiene toda la información que define un comentario en la aplicación como es el autor, es decir, el usuario que lo escribe, la fecha en que se realiza o la fecha de última modificación.
- Clase Usuarios: en ella se encuentran todos los métodos relacionados con la gestión última sobre los distintos usuarios de nuestra aplicación, como por ejemplo, eliminarUsuario().
- Clase Eventos: engloba todas los métodos relacionados directamente con la gestión del conjunto de los eventos de nuestra aplicación, como por ejemplo el método filtrarEventos().
- Clase Comentarios: contiene todos los métodos necesarios para la elaboración, modificación, eliminación y cualquier tipo de gestión relacionada con los comentarios.

Vista

Aquí definimos la parte visual del sistema, es decir, donde establecemos todas la interfaces gráficas que permiten la interacción entra la aplicación y el cliente. Para ello disponemos de la siguiente clase:

- Clase View: en ella se concentran todos los métodos necesarios para representar gráficamente nuestra aplicación. La terminal (el medio gráfico que usaremos para la interacción entre el usuario y la aplicación) variará según las opciones que elija el usuario. Nótese que habrá diferencia en la interfaz en cada uno de los métodos. En el método mostrarMenu() se representa gráficamente la primera imagen de nuestra aplicación. En ella el usuario verá un menú enumerado con las siguientes opciones: 1.Iniciar Sesión, 2.Registrarse, 3.Buscar eventos y 4.Salir de la aplicación. Según el número que escriba el usuario en la terminal se cargarán otros métodos mostrando otras interfaces. Si el usuario elige la opción 3 se cargará el método mostrarEventos(), el cual mostrará los eventos de 10 en 10, priorizando los más recientes y los patrocinados, los cuales tendrán asegurados y ocuparán los tres primeros puestos en la lista de los eventos mostrados. Si el usuario elije la opción 4, el programa se cerrará por completo y no se mostrará ninguna interfaz más. Si el usuario elije la opción 2 se cargará el método mostrarLogin(), el cual representará una interfaz totalmente distinta, ya que no mostrará ningún menú sino que simplemente se mostrará una frase pidiendo al usuario que escriba un nombre de usuario correcto (“Introduzca su nombre de usuario”) y posteriormente, si ésta es correcta, mostrará otro mensaje pidiendo una contraseña válida (“Introduzca su contraseña”). Si el usuario ha introducido la información correctamente se irá pidiendo poco a poco información relevante sobre el usuario, como el nombre, los apellidos o el sexo. Una vez finalizado este proceso se cargará otro método para visualizar la siguiente interfaz. Cabe destacar que durante la creación del perfil se mostrará un mensaje de error por cada uno de los datos que haya introducido de manera errónea: (“¡El nombre de usuario que ha elegido no está disponible, te damos otra oportunidad, se más original esta vez!”) si introduce mal el nombre de usuario o (“Contraseña inválida, recuerda que debe tener un mínimo de 6 caracteres y al menos una letra en mayúscula. Inténtalo de nuevo ;)”) si introduce mal la contraseña.

  Si el usuario introduce correctamente esta información se cargará la siguiente interfaz donde se mostrará una lista enumerada con las opciones que puede realizar el usuario: 1.Crear evento, 2.Buscar evento, 3.Buscar usuarios, 4.Ver

perfil y 5.Salir de la aplicación. Si el usuario elige la opción 4 se mostrará la información del usuario que disponga la aplicación y una lista enumerada con las mismas opciones anteriores, exceptuando la primera "Ver Perfil". En su puesto se encontrará la opción de modificar el perfil. Si elige esta opción, se mostrarán poco a poco preguntas como las siguientes para recoger la información del usuario: ("Introduce tu nombre…", "…y apellidos", "¿Cuál es el día de tu fecha de nacimiento?", "¿Dónde vives?", "¿Cuál es tu estado civil?", "¿Eres hombre o mujer?", "Bueno, no seamos tan formales…más bien…escribe tu frase favorita :) "). Cabe destacar que la elegir la opción 4 (ver perfil) se mostrará de manera aleatoria un dibujo ASCII sobre un animal. Las opciones del animal son: 1. Pez, 2.Pájaro, 3.Cerdo, 4.Oso, 5. Rana. Por cada pregunta se esperará a que el usuario introduzca la información deseada para dar el salto a la próxima pregunta, hasta que sea la última, en ese caso se cargará de nuevo la interfaz anterior. Si el usuario eligiese la opción 2, se mostrarán enumerados de 10 en 10 los distintos eventos que se encuentren en la aplicación. Lo mismo sucedería si eligiese la opción 3, es decir, se mostraría por pantalla la información de todos los usuarios registrados en la aplicación.

  Finalmente, si el usuario elige la opción 5, la aplicación se cerrará de manera inmediata. Si entre la información del usuario aparece que el sexo del mismo es masculino, las letras se representarán en color amarillo. Por el contrario, si es femenino o no ha especificado el sexo, se mostrará en rojo claro. En el resto de la aplicación predominará el color verde ya que gran parte de los mensajes se mostrarán en este color, exceptuando algunos dibujos coloridos a modo de decoración o mensajes de error, que se mostrarán en rojo.
- Clase Errores: se ha creado esta clase que contiene los distintos errores por comodidad, es decir, para acceder de manera más fácil a los mismos.

Controlador

Esta parte del patrón es la que establece conexión entre Modelo y Vista. Incluye las siguientes clases:

- Clase Main: esta clase contiene el método main que ejecuta la aplicación y crea las instancias de las interfaces gráficas principales de la aplicación, como es el caso del método para mostrar la pantalla principal de la aplicación, mostrarMenu().
- Clase Controller: esta clase contiene toda la lógica de relaciones entre el modelo y las vistas, es decir, entre la información que maneja la aplicación y lo que se muestra al usuario. Esta clase contiene todos los métodos que permiten al usuario realizar todas las actividades posibles en nuestra aplicación como es: registrarse, iniciar sesión en la aplicación, comentar un evento o un perfil de otro usuario, crear, eliminar o modificar un evento, modificar tu propio perfil, eliminar tu cuenta o filtrar los eventos o los usuarios para facilitar su búsqueda.

5. Diseño detallado del sistema

El sistema dispondrá de tres paquetes, diez clases y seis ficheros.

Paquetes: los paquetes correspondientes a Modelo-Vista-Controlados. El paquete Controlador agrupará las clases Main y Controller, el paquete Vista agrupara a las clases View y Errores y el paquete Model agrupará a las clases Usuario, Usuarios, Comentario, Comentarios, Evento, Eventos y Model.

Clases: Main, Model, View, Errores, Controler, Usuarios, Eventos, Comentarios, Usuario, Evento, Comentario.

Ficheros: habrá un fichero que guarde a los usuarios registrados, otro con los eventos normales, otro con los eventos promocionados y los otros tres ficheros serán copias de cada uno de los anteriores ficheros. El administrador realizará las copias semanalmente.

El programa seguirá el modelo de Model-View-Controller.

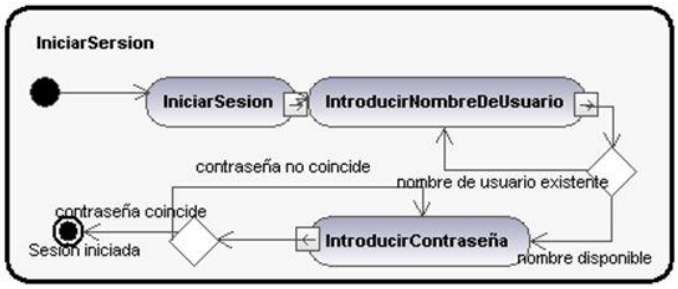
Siguiendo este modelo, la clase Main estará relacionada con la clase Controller. En la clase Main se ejecutara un bucle que ira ejecutando las diferentes funciones de Controller según el usuario vaya moviéndose por el programa.

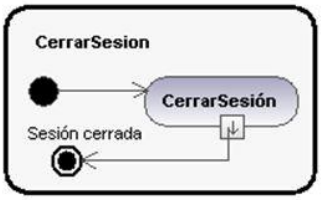
En Main sólo será necesaria la creación de un objeto Controller. Además de otros objetos como dos de Usuarios y uno de Usuario.

Controller está asociado con View y con Model. A través de View podrá mostrar por pantalla los diferentes menús y avisos, y a través de Model podrá realizar las acciones que conciernen a la opción seleccionada.

Por último, Model tiene asociación con Usuarios y Eventos.

Desde la clase Model se pueden realizar acciones elementales como IniciarSesión o CerrarSesión:





Para la acción de iniciar sesión se ha de acceder a la clase Usuarios para verificar los datos.

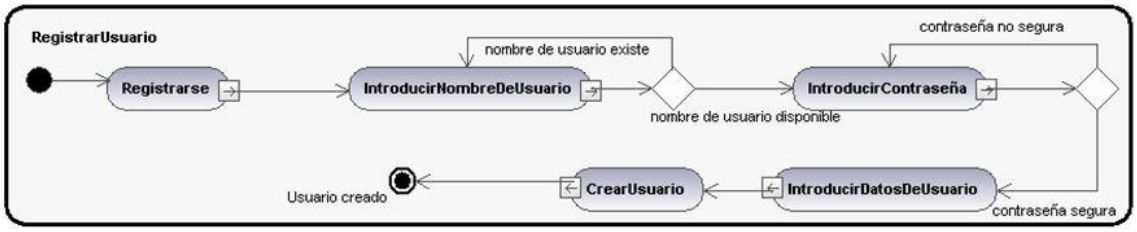
Usuarios y Eventos son dos clases que agrupan un conjunto de objetos tipo Usuario o Evento, y permiten su organización y modelado. De este modo a través de estas clases

se pueden realizar todas las acciones posibles sobre los distintos tipos de usuarios o eventos.

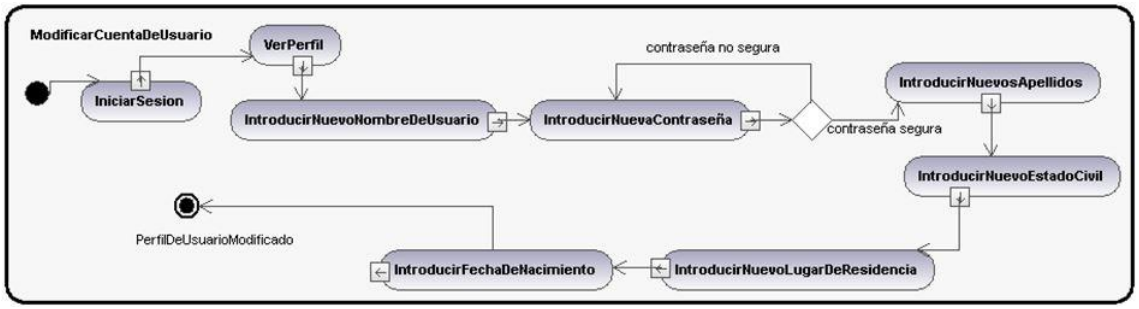
Usuarios tiene asociación de composición con Usuario y Eventos tiene asociación de composición con Evento.

Desde Usuarios puedes generar objetos Usuarios con objetos Usuario determinados (por ejemplo filtrados). También puedes devolver un objeto Usuario específico determinado, eliminar una cuenta de usuario o modificar la información de un usuario. Así mismo también puede realizar acciones como seguir usuarios.

Por lo tanto, desde la clase Usuarios se pueden realizar acciones tales como RegistrarUsuario:

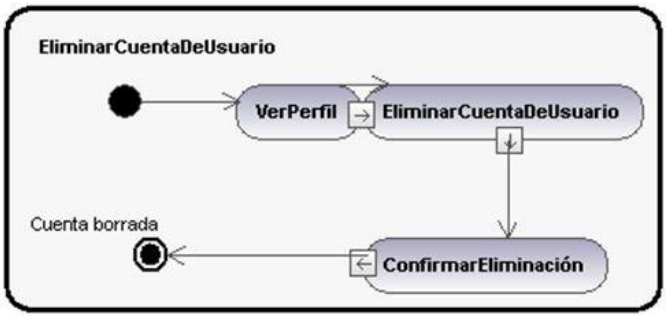


ModificarCuentaDeUsuario:



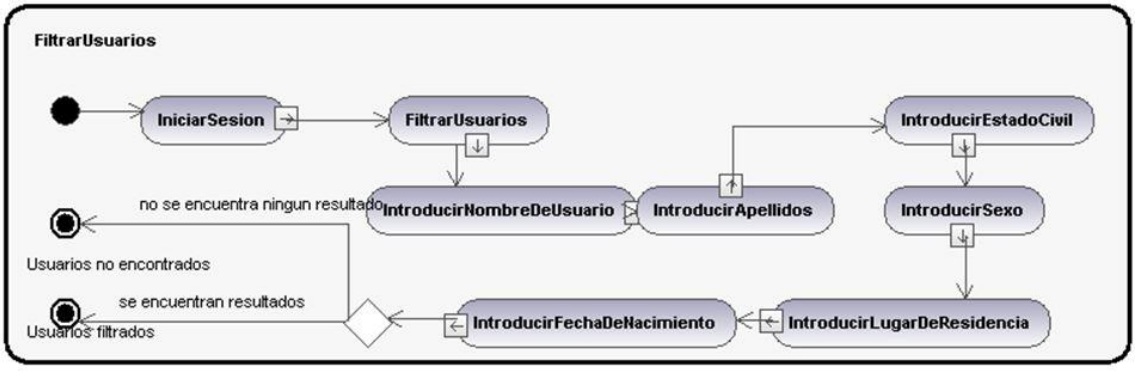
Para llevar a cabo la modificación de un objeto Usuario, Usuarios ha de acceder a la clase Usuario.

EliminarCuentaDeUsuario:



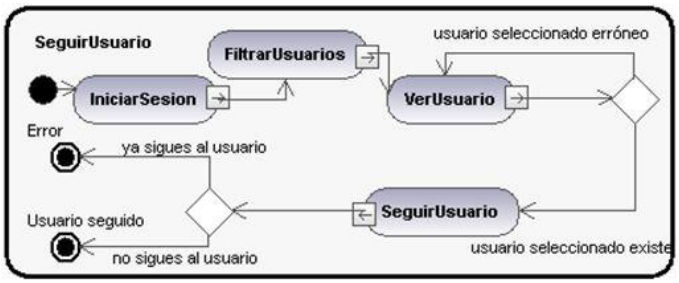
Para eliminar una cuenta de usuario, se ha de acceder la clase Usuario, para verificar que se está borrando el usuario que ha iniciado sesión en el sistema.

FiltrarUsuarios:



Para filtrar usuarios se ha de acceder a la clase Usuario para ir comprobando los diferentes campos que se quieren filtrar.

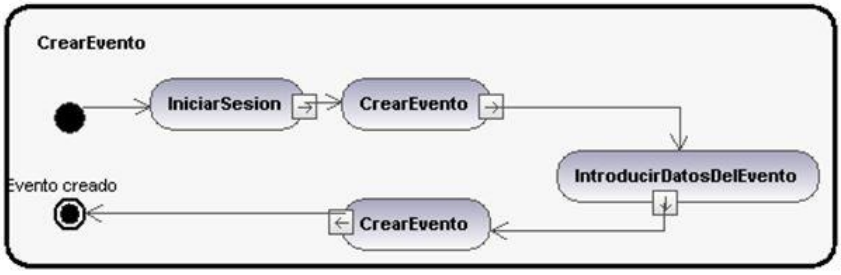
O SeguirUsuario:



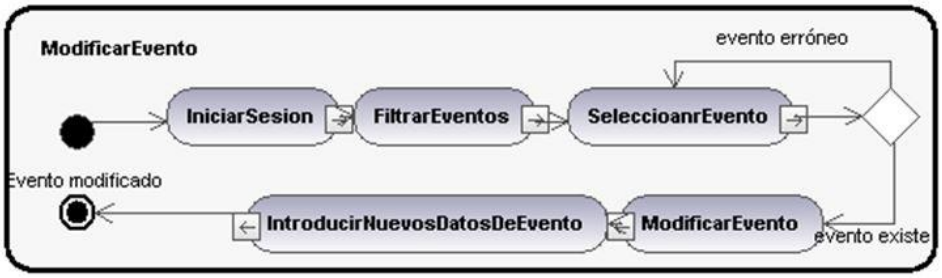
Para seguir a un usuario se ha de acceder a la clase Usuario para obtener datos como el usuario al que se va a seguir y el usuario que va a seguir a ese usuario.

El comportamiento de la clase Usuarios sucede de forma análoga con Eventos. Por esta razón se puede reciclar mucho código, lo cual simplifica la implementación del sistema.

Por lo tanto, desde la clase Eventoss se pueden realizar acciones tales como CrearEvento:

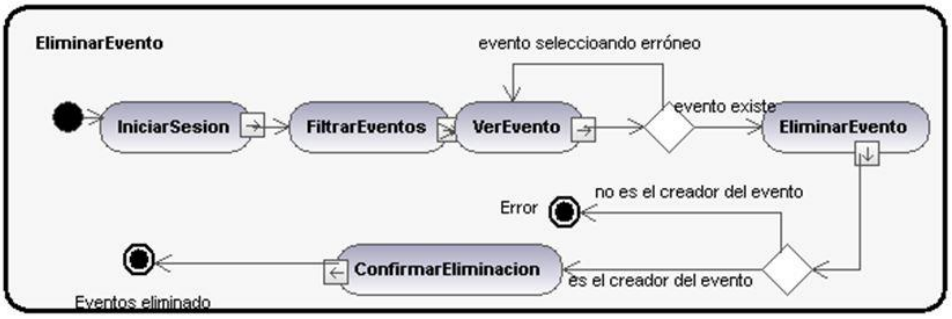


ModificarEvento:



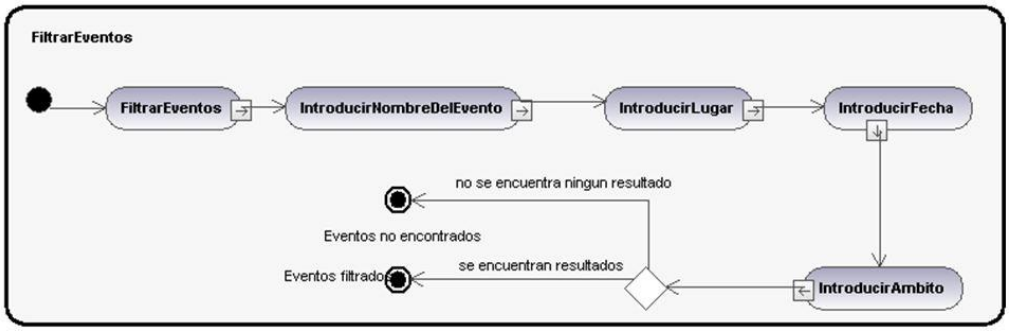
Para llevar a cabo la modificación de un objeto Usuario, Usuarios ha de acceder a la clase Evento.

EliminarEvento:



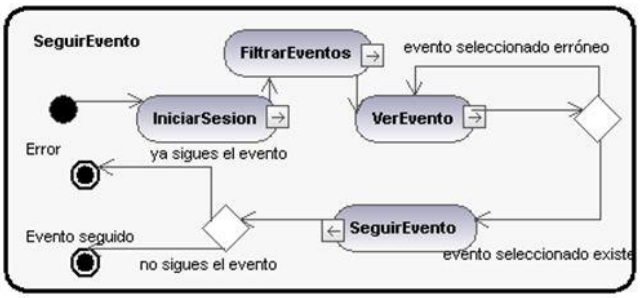
Para eliminar un evento, se ha de acceder la clase Usuario, para verificar que se está borrando un evento creado por el usuario que quiere borrar dicho evento.

FiltrarEventos:



Para filtrar eventos se ha de acceder a la clase Evento para ir comprobando los diferentes campos que se quieren filtrar.

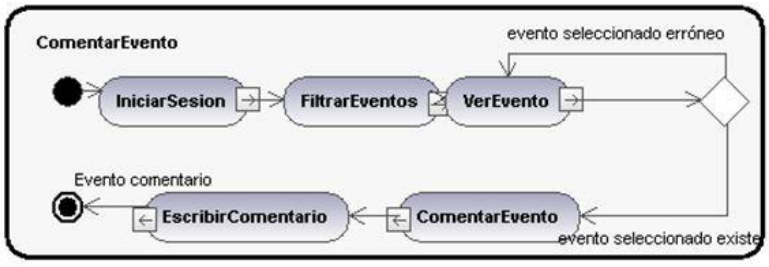
O SeguirEvento:



Para seguir un evento se ha de acceder a la clase Usuario y Evento para obtener datos como el evento que se va a seguir y el usuario que va a seguir a ese evento.

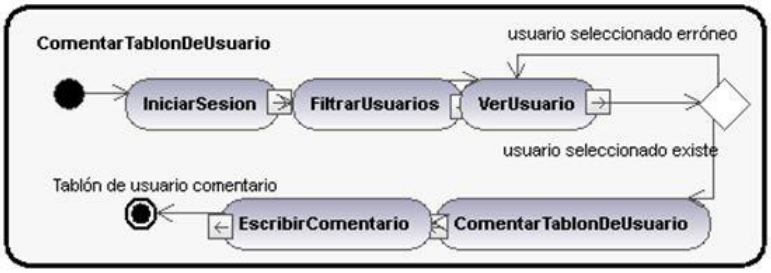
Por último, tanto la clase Usuario como la clase Evento tienen asociación con Comentarios. El funcionamiento es el mismo que el explicado anteriormente.

Por lo que desde Comentarios podemos realizar acciones como ComentarEvento:



Para realizar esta acción se accede a las clases Usuario y Evento para obtener los datos del usuario que comenta el evento y modificar el objeto Comentarios del evento.

O ComentarTablonDeUsuario:



Para esta acción se accede a la clase Usuario, para obtener los datos del usuario que comenta en el tablón del otro usuario, y para modificar el objeto Comentarios del usuario al que se comenta.

Por lo tanto tenemos:

Main: raíz del sistema. Controla el orden de ejecución de las diferentes partes del programa. La implementación óptima sería un bucle while cuya ejecución será continua hasta la salida del programa. Dentro un switch en el que cada caso represente una parte del programa (registrar usuario, filtrar eventos, iniciar sesión, etc…) cada caso tiene su método correspondiente en la clase Controller. Estos métodos devuelven el valor a la próxima parte del programa por lo que los métodos determinan a dónde irá el programa y la clase Main ejecuta ese orden. El bucle while hace que el movimiento por el programa pueda ser libre.

Controller: interacciona con View y con Model para realizar las diferentes acciones del sistema. Contiene las diferentes partes del programa y se encarga de que realicen bien sus objetivos. Desde los métodos de la clase Controller se interactuará con la clase View y la clase Model para que todo lo referente a una parte del programa se englobe en dicho método. Cada método deberá devolver un valor que le permita a Main determinar el siguiente método a ejecutar. En algunos casos estos métodos devolverán objetos tipo Usuario, Usuarios o Eventos. En estos casos este objeto cumplirá la anterior función además de proporcionarle a Main el objeto Usuario o Usuarios o Eventos.

View: representa por pantalla los diferentes menús y advertencias. Se encarga de la parte gráfica del sistema. Sus objetos son creados desde la clase Controller. Su función principal es proporcionar los métodos necesarios para representar los diferentes menús, eventos, usuarios… del sistema.

Errores: representa por pantalla todos los posibles errores que se puedan producir durante la ejecución del programa. El método de mostrarError() deberá recibir como parámetro un int y dependiendo del valor de ese int representará un error u otro.

Model: realiza las operaciones dentro de una acción. Encargado de llamar a las modificaciones de Usuarios, Eventos y Comentarios. Sus objetos son creados desde la clase Controller. Para su correcto funcionamiento, los métodos de la clase Model deben realizar todas las acciones que no incluyan la parte gráfica dentro de las modificaciones,

creaciones o eliminaciones de eventos o usuarios. Para cumplir con esta función ha de crear diferentes objetos tipo Usuario, Usuarios, Evento y Eventos y realizar sus modificaciones mediante llamadas a sus métodos.

Eventos: contiene objetos Evento. Permite el filtraje, localización y modificación de los diferentes objetos Evento. Trata de realizar todas las acciones posibles sobre los eventos sin que la clase Model tenga que acceder a objetos de la clase Evento. Por lo tanto Eventos crea objetos tipo Evento y realiza modificaciones individuales además de acciones más generales como llamar a métodos que generen devuelvan nuevos objetos tipo Evento pero con determinados objetos tipo Evento (filtrados por ejemplo).

Evento: objeto que reúne toda la información que hace referencia a un evento determinado. Permite su creación y modificación. Su función principal es servir de nodo para la clase Eventos.

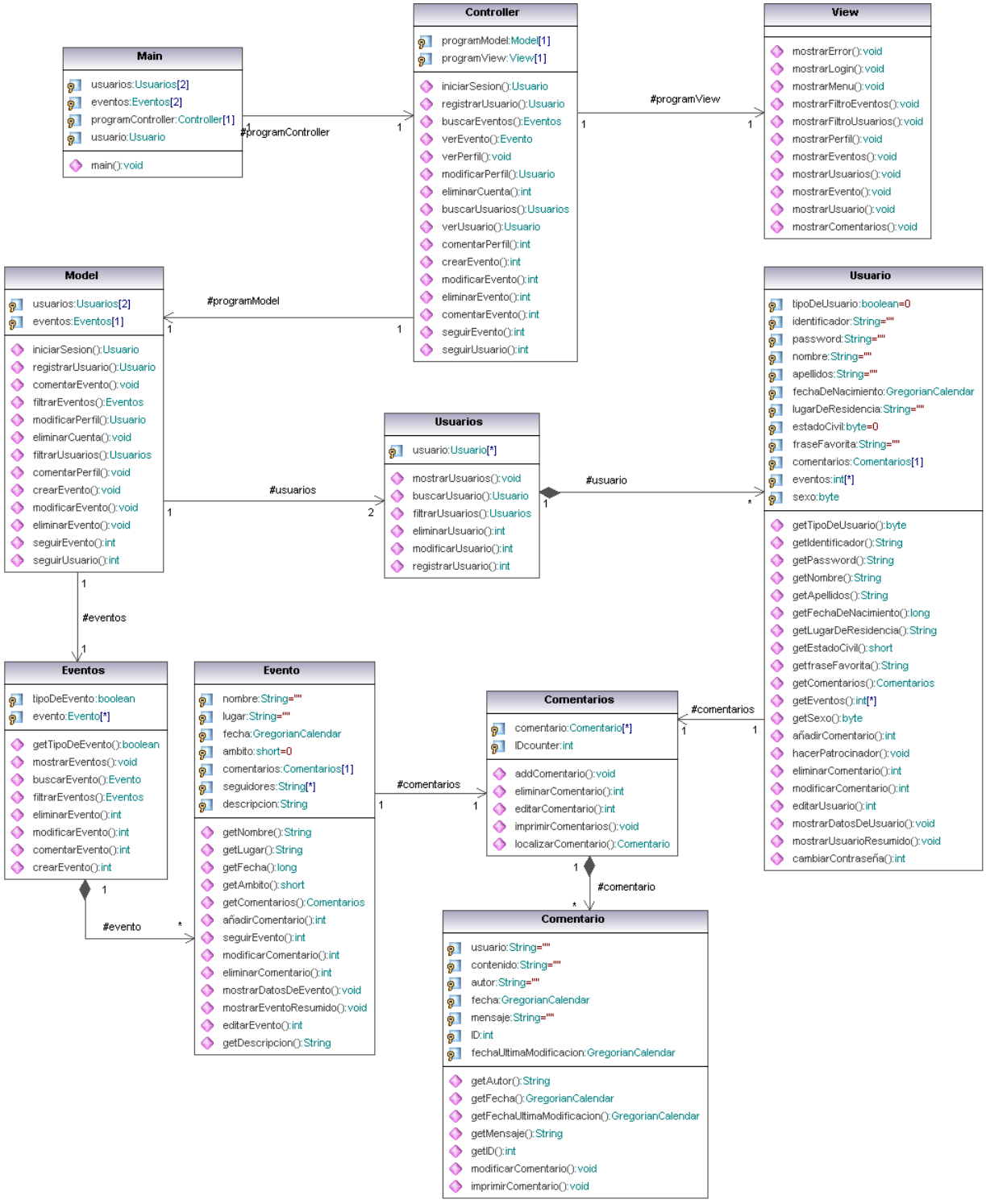
Usuarios: contiene objetos Usuario. Permite el filtraje, localización y modificación de los diferentes objetos Usuario. Trata de realizar todas las acciones posibles sobre los usuarios sin que la clase Model tenga que acceder a objetos de la clase Usuario. Por lo tanto Usuarios crea objetos tipo Usuario y realiza modificaciones individuales además de acciones más generales como llamar a métodos que generen devuelvan nuevos objetos tipo Usuario pero con determinados objetos tipo Usuario (filtrados por ejemplo).

Usuario: objeto que reúne toda la información que hace referencia a un usuario determinado. Permite su creación y modificación. Su función principal es servir de nodo para la clase Usuario. Cabe destacar su función en clases superiores como Main o Controller ya que el objeto Usuario correspondiente a la persona que está ejecutando el programa tiene presencia en varias partes del programa.

Comentarios: contiene objetos Comentario. Permite el creación y modificación de los diferentes objetos Comentario. Trata de realizar todas las acciones posibles sobre los comentarios sin que la clase Model tenga que acceder a objetos de la clase Comentario. Por lo tanto Comentarios crea objetos tipo Comentario y realiza modificaciones individuales.

Comentario: objeto que reúne toda la información que hace referencia a un comentario determinado. Permite su creación y modificación. Su función principal es servir de nodo para la clase Comentarios.

A continuación se muestran el diagrama de clases del programa:


Main
usuarios:Usuarios[2]
eventos:Eventos[2]
programController:Controller[1]
usuario:Usuario
main():void
#programController
Controller
programModel:Model[1]
programView:View[1]
iniciarSesion():Usuario
registrarUsuario():Usuario
buscarEventos():Eventos
verEvento():Evento
verPerfil():void
modificarPerfil():Usuario
eliminarCuenta():int
buscarUsuarios():Usuarios
verUsuario():Usuario
comentarPerfil():int
crearEvento():int
modificarEvento():int
eliminarEvento():int
comentarEvento():int
seguirEvento():int
seguirUsuario():int
#programView
View
mostrarError():void
mostrarLogin():void
mostrarMenu():void
mostrarFiltroEventos():void
mostrarFiltroUsuarios():void
mostrarPerfil():void
mostrarEventos():void
mostrarUsuarios():void
mostrarEvento():void
mostrarUsuario():void
mostrarComentarios():void
Model
usuarios:Usuarios[2]
eventos:Eventos[1]
iniciarSesion():Usuario
registrarUsuario():Usuario
comentarEvento():void
filtrarEventos():Eventos
modificarPerfil():Usuario
eliminarCuenta():void
filtrarUsuarios():Usuarios
comentarPerfil():void
crearEvento():void
modificarEvento():void
eliminarEvento():void
seguirEvento():int
seguirUsuario():int
#programModel
#usuarios
#eventos
Usuarios
usuario:Usuario[*]
mostrarUsuarios():void
buscarUsuario():Usuario
filtrarUsuarios():Usuarios
eliminarUsuario():int
modificarUsuario():int
registrarUsuario():int
#usuario
Usuario
tipoDeUsuario:boolean=0
identificador:String=""
password:String=""
nombre:String=""
apellidos:String=""
fechaDeNacimiento:GregorianCalendar
lugarDeResidencia:String=""
estadoCivil:byte=0
fraseFavorita:String=""
comentarios:Comentarios[1]
eventos:int[*]
sexo:byte
getTipoDeUsuario():byte
getIdentificador():String
getPassword():String
getNombre():String
getApellidos():String
getFechaDeNacimiento():long
getLugarDeResidencia():String
getEstadoCivil():short
getfraseFavorita():String
getComentarios():Comentarios
getEventos():int[*]
getSexo():byte
añadirComentario():int
hacerPatrocinador():void
eliminarComentario():int
modificarComentario():int
editarUsuario():int
mostrarDatosDeUsuario():void
mostrarUsuarioResumido():void
cambiarContraseña():int
Eventos
tipoDeEvento:boolean
evento:Evento[*]
getTipoDeEvento():boolean
mostrarEventos():void
buscarEvento():Evento
filtrarEventos():Eventos
eliminarEvento():int
modificarEvento():int
comentarEvento():int
crearEvento():int
#evento
Evento
nombre:String=""
lugar:String=""
fecha:GregorianCalendar
ambito:short=0
comentarios:Comentarios[1]
seguidores:String[*]
descripcion:String
getNombre():String
getLugar():String
getFecha():long
getAmbito():short
getComentarios():Comentarios
añadirComentario():int
seguirEvento():int
modificarComentario():int
eliminarComentario():int
mostrarDatosDeEvento():void
mostrarEventoResumido():void
editarEvento():int
getDescripcion():String
#comentarios
Comentarios
comentario:Comentario[*]
IDcounter:int
addComentario():void
eliminarComentario():int
editarComentario():int
imprimirComentarios():void
localizarComentario():Comentario
#comentarios
#comentario
Comentario
usuario:String=""
contenido:String=""
autor:String=""
fecha:GregorianCalendar
mensaje:String=""
ID:int
fechaUltimaModificacion:GregorianCalendar
getAutor():String
getFecha():GregorianCalendar
getFechaUltimaModificacion():GregorianCalendar
getMensaje():String
getID():int
modificarComentario():void
imprimirComentario():void

6. Plan de pruebas

6.1 Lo que se va a probar

Comprobar que en el caso de introducir un nombre de usuario que existe el sistema lo rechaza y avisa al usuario.

Comprobar que el perfil de usuario se almacena de manera correcta en el repositorio.

Comprobar que la función de crear el perfil comprueba la longitud de la contraseña y si tiene una letra mayúscula.

Comprobar que el sistema puede encontrar el perfil dado el nombre de usuario y la contraseña correcta.

Comprobar que ambos tipos de usuarios pueden visualizar la lista de eventos

Comprobar que el usuario esta capaz de modificar su perfil con éxito

Comprobar que el usuario esta capaz de eliminar su perfil con éxito

Comprobar que los eventos se visualizan de la manera especificada en los requisitos funcionales.

Comprobar que los usuarios registrados pueden seguir otros usuarios registrados

Comprobar que un usuario registrado puede crear eventos con éxito.

Comprobar que un evento patrocinado se almacena en la lista de eventos patrocinados

Comprobar que el creador de un evento sea el único a poder modificarlo

Comprobar que el creador de un evento sea el único a poder eliminarlo

Comprobar que un usuario registrado puede inscribirse en un evento

Comprobar que un usuario puede buscar por filtros y que consigue resultados correctos.

Comprobar que el usuario registrado puede modificar su cuenta.

Comprobar que un usuario registrado puede eliminar su cuenta y que es el único que puede hacerlo

Comprobar que se puede hacer comentarios a los eventos y que se muestran correctamente.

Comprobar que los usuarios registrados pueden comentar en los perfiles de otros usuarios registrados.

6.2 Lo que no se va a comprobar:

No se va a comprobar la seguridad del sistema.

No se va a comprobar la facilidad de uso del sistema.

No se va a comprobar el rendimiento en tiempo del sistema.

No se va a comprobar el tamaño que va a ocupar el sistema .

7. Glosario

- **ASCII:** Código estándar Estadounidense para el intercambio de información.
- **Clase (Java):** abstracción que define un tipo de objeto especificando qué propiedades y operaciones disponibles va a tener.
- **DDBB / Base de Datos:** conjunto de datos pertenecientes a un mismo contexto y almacenados sistemáticamente para su posterior uso.
- **Depuración:** revisar y analizar si la sintaxis de un programa creado es correcta y/o genera errores al ejecutarlo.
- **ERS:** documento de Especificación de Requisitos de Software (Software Requirements Specification, SRS).
- **Hardware:** elementos físicos o materiales que constituyen una computadora o un sistema informático.
- **IEEE:** es el Instituto de Ingenieros Eléctricos y Electrónicos, una asociación técnico-profesional mundial dedicado a la estandarización.
- **Interfaz (usuario):** destinada a entregar información acerca de los procesos y herramientas de control, a través de esta, el usuario es capaz de comunicarse con el sistema.
- **Login:** es el proceso mediante el cual se controla el acceso individual a un sistema informático mediante la identificación del usuario utilizando credenciales provistas por el usuario.
- **Main:** clase principal de un programa Java.
- **Método (Java):** conjunto de instrucciones definidas dentro de una clase, que realizan una determinada tarea y a las que podemos invocar mediante un nombre.
- **MVC:** patrón de arquitectura de software que separa los datos y la lógica de negocio de una aplicación de la interfaz de usuario y el módulo encargado de gestionar los eventos y las comunicaciones (Modelo-Vista-Controlador).
- **Objeto (Java):** entidad existente en la memoria del ordenador que tiene unas propiedades (atributos o datos sobre sí mismo almacenados por el objeto) y unas operaciones disponibles específicas (métodos).
- **Persistencia:** es la acción de preservar la información de un objeto de forma permanente (guardado), pero a su vez también se refiere a poder recuperar la información del mismo (leerlo) para que pueda ser nuevamente utiliza.
- **Registro / Registrar:** acción de introducir los datos en el sistema.
- **Repositorio:** es un sitio centralizado donde se almacena y mantiene información digital, habitualmente bases de datos o archivos informáticos.
- **SDD:** documento de desarrollo sobre la descripción de un producto de software (Software Design Document).
- **SO:** Sistema Operativo, es un programa o conjunto de programas de un sistema informático que gestiona los recursos de hardware y provee servicios a los programas de aplicación.
- **Software:** conjunto de programas y rutinas que permiten a la computadora realizar determinadas tareas.
- **SPMP:** documento con el plan de gestión de un proyecto software (Software Project Management Plan).
- Terminal /Consola de comandos: interfaz (no gráfica) para comunicarse con un

computador, la entrada de datos se realiza mediante teclado y se visualiza en la pantalla. Solo se muestran caracteres alfanuméricos.

8. Referencias:

[1]. Documento SPMP: Documento entregado el día 8 de octubre. Contiene información general sobre la gestión del proyecto. **Recomendada su lectura previa.**
[2]. Documento ERS: Documento entregado el día 8 de octubre. Contiene información sobre los requisitos del proyecto. **Recomendada su lectura previa.**
[3]. IEEE1016: Recomendación sobre el desarrollo de un documento de diseño elaborado por la IEEE. Accesible de modo gratuito en este enlace: https://aulaglobal.uc3m.es/pluginfile.php/1317477/mod_page/content/6/ieee1016.pdf