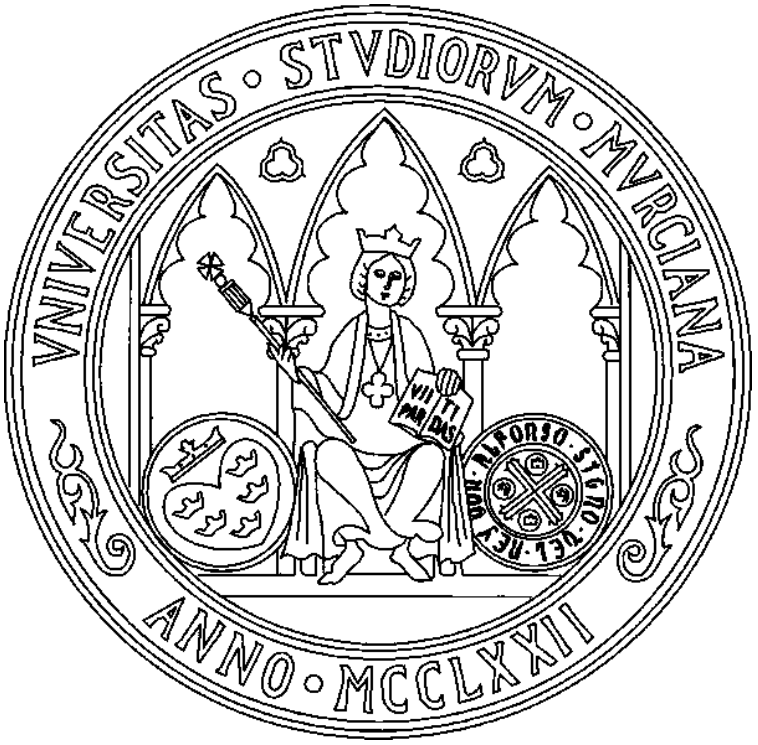
TECNOLOGÍAS DE DESARROLLO SOFTWARE

APPCHAT



UNIVERSIDAD DE MURCIA
GRADO EN INGENIERÍA INFORMÁTICA
CURSO 2019/2020

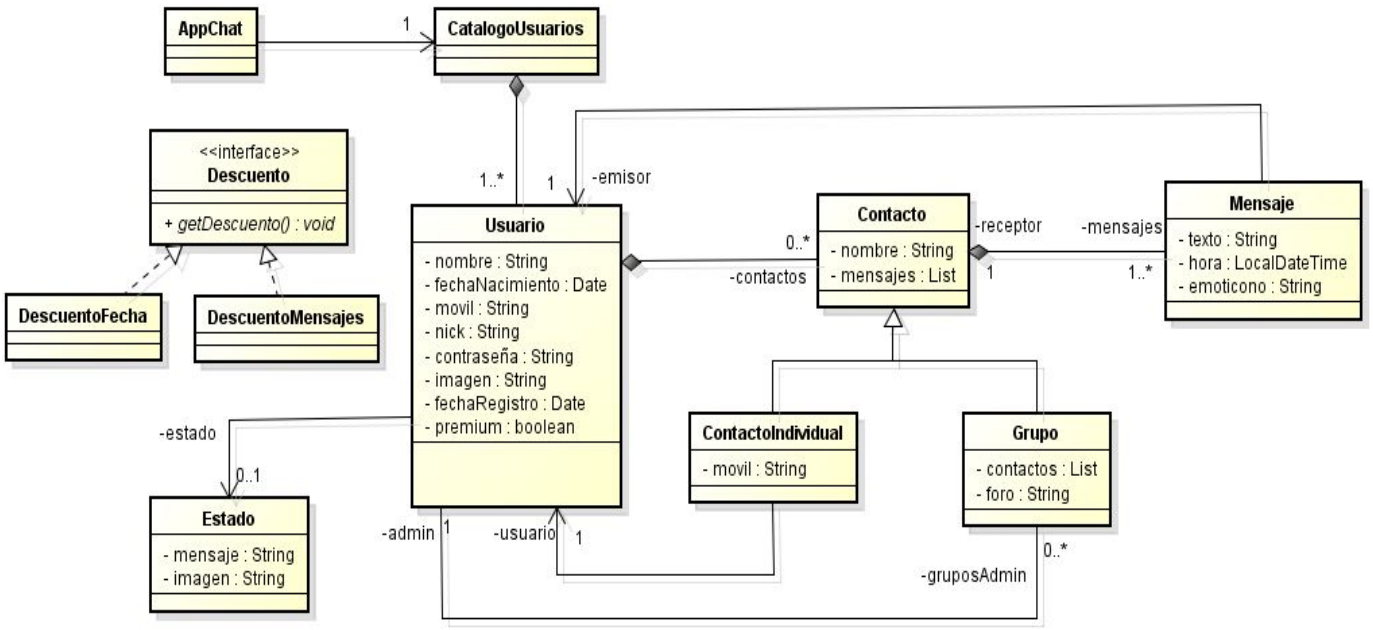
PROFESOR : FRANCISCO JAVIER BERMÚDEZ RUIZ.

ALUMNOS :

- GONZÁLEZ ALCARAZ, LUCIA. 49277694X. G2.2. lucia.gonzaleza@um.es
- ORTIZ ARAGÓN, JAIME. 49196689B. G2.2. jaime.ortiza@um.es

ÍNDICE

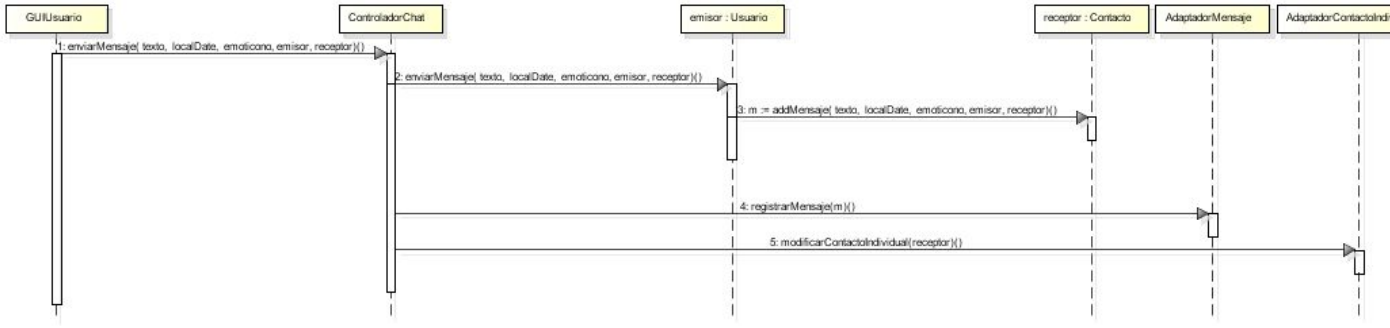
1.DIAGRAMA DE CLASES DE DOMINIO



Este es el diagrama de clases que hemos seguido para la implementación del proyecto, muestra las relaciones más relevantes entre las distintas clases del modelo, así faltarían algunas clases de poca relevancia como por ejemplo las empleadas para proporcionar la información de uso al usuario cuando es premium. Vamos a explicar un poco las relaciones que se muestran.

Como se puede observar en el diagrama un usuario contendrá varios contactos que podrán ser contactos individuales o grupos. Un mensaje tendrá como emisor a un usuario y como receptor a un contacto. Finalmente un usuario podrá hacerse premium y obtener algún descuento según su fecha de registro o el número de mensajes.

2. DIAGRAMA DE SECUENCIA PARA LA OPERACIÓN DE ENVIAR UN MENSAJE A UN CONTACTO INDIVIDUAL.

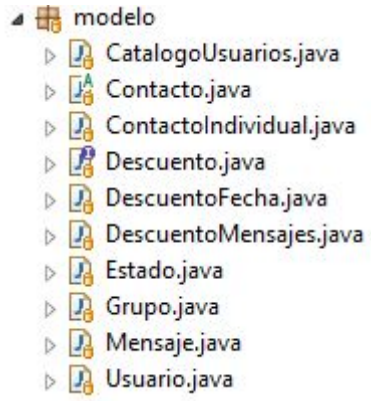


El usuario desde la ventana *NuevaVentanaChat* pulsará en el contacto al que quiere enviar en mensaje, lo que abrirá su chat *Chats*. En este panel escribirá en mensaje en el área de texto y le dará a enviar. En este momento se llama al controlador con *enviarMensaje*, donde se le pasa los datos necesarios para la creación de un mensaje como son el texto, la fecha, el emoticono, el emisor y el receptor. Desde la clase *Controlador*, llamamos al modelo en la clase Usuario que a su vez llama a la clase Contacto donde crea el mensaje y lo añade a su lista de mensajes.

Por último, para no crear inconsistencias,registramos el mensaje y actualizamos el contacto del receptor, ya que su lista de mensajes ha sido actualizada.

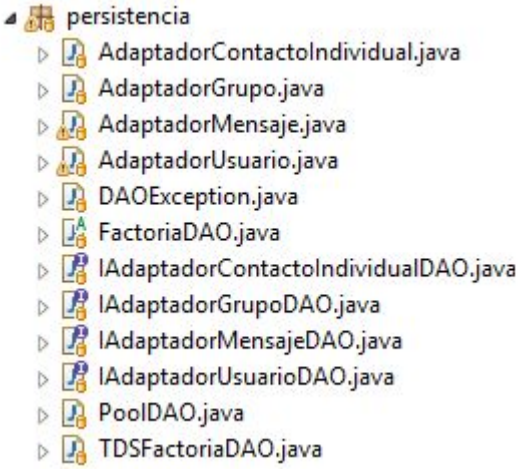
3. EXPLICACIÓN DE LA ARQUITECTURA DE LA APLICACIÓN Y DECISIONES DE DISEÑO MÁS RELEVANTES

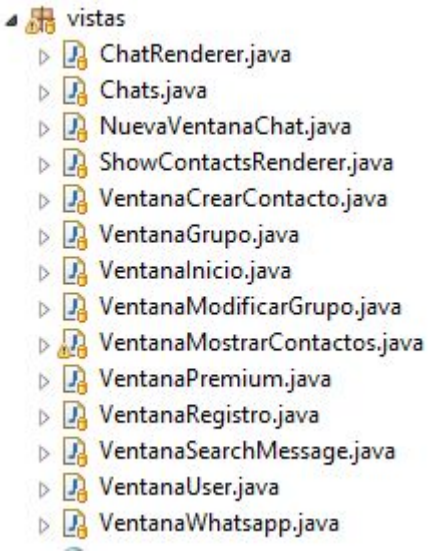
Para el desarrollo de la aplicación hemos utilizado el patrón de arquitectura de 3 capas para diferenciar claramente cada parte y facilitar la actualización y el mantenimiento futuro. Esta división sigue el principio estudiado en teoría de "Modelo - Vista", que consiste en implementar una capa de dominio, una capa de persistencia que trate los aspectos más técnicos y una capa de presentación con la que interactúa directamente el usuario. Las tres capas claramente diferenciadas en nuestro proyecto son modelo, persistencia y vistas.



La imagen anterior muestra las clases que pertenecen a la capa del **modelo**, estas clases engloban la lógica del proyecto. La relación entre ellas la podemos observar en el diagrama de clases.

A la izquierda mostramos la capa de **persistencia** la cual utiliza un patrón DAO. Se ha utilizado la base de datos H2, que se nos proporcionó por parte del profesorado.





Por último tenemos la capa de **vistas**, la encargada de interactuar con el cliente. Este paquete incluye las clases implementadas en JSwing que actúan de GUI para la interacción con el usuario.

Finalmente para conectar estas capas tenemos un **controlador** que hace de fachada entre modelo, persistencia y vistas. Esta clase relaciona la lógica de negocio de modelo con la GUI con el fin de no incurrir en incumplimiento de patrones de los que hablaremos más adelante.

4. EXPLICACIÓN DE LOS PATRONES UTILIZADOS

El patrón principal es el **patrón Singleton** utilizado para asegurarnos de la única instancia del controlador con el fin de acceder a los adaptadores. Al final de la implementación de la práctica también nos vimos forzados a usarlo para para proporcionar el acceso unificado al catálogo de usuario.En un principio, nuestra lógica consistía en ir pasando como parámetro el objeto Usuario que está trabajando en ese momento, pero al final, para la implementación del componente, nos dimos cuenta de que teníamos que recuperar el usuario sin poder pasarlo como parámetro, ya que estas clases vienen implementadas por la asignatura.

El **patrón adaptador** es utilizado en los distintos adaptadores mencionados en la capa de persistencia. El objetivo principal es tratar los objetos que almacenamos en la base de datos para poder recuperarlos, modificarlos, eliminarlos, etc.

Otro patrón utilizado es el **patrón factoría** para generar todos los tipos de adaptadores de nuestra práctica. La idea es que cada subclase se pueda instanciar.

El **patrón DAO** se utiliza para el almacenamiento de datos en la base de datos, y que este proceso sea independiente del sistema de persistencia que se use.

En nuestro caso el controlador hace de **patrón fachada** ya que se encarga de enlazar las diferentes capas. Ligado a esto encontramos el patrón **experto**. Como se ha mencionado el el apartado anterior, la arquitectura de la aplicación se divide en capas y el controlador actúa de experto con el fin de adherir unas capas con otras. Esto además proporciona un bajo acoplamiento, al no ir construyendo la lógica de la aplicación encima de una base, sino que todas las partes funcionan por separado.

Finalmente nuestra implementación de los descuentos para hacerse premium, podría verse como un **patrón estrategia**, al coexistir dos opciones en función de la fecha de registro o el número de mensajes mandados en la aplicación.

5. EXPLICACIÓN DE LOS COMPONENTES UTILIZADOS

En el desarrollo de la práctica se han empleado diversos componentes que detallamos a continuación.

- **jcalendar**: para mostrar un calendario en la ventana de registro.

- **jTattoo**: para el diseño de las ventanas del chat. No presenta una funcionalidad real para la aplicación más que la apariencia.

- **ChatWindowLib**: para la creación de burbujas de los mensajes.

- **Xchart**: para la generación de diagramas en la información de uso de un usuario.

- **iText**: para la generación de un documento pdf en la funcionalidad que tienen los usuarios de información de uso.

- **Luz**: empleado para la creación de nuestro componente del cargador de mensajes. Este componente se nos proporciona implementado por parte de la asignatura y lo hemos puesto en la ventana principal del chat (*NuevaVentanaChat*) complementando al implementado por nosotros que se explica a continuación.

- **cargador**: componente de creación propia que sirve para importar mensajes whatsApp a nuestro chat. Este componente consta de 3 paquetes para cargar las listas de mensajes, añadir los listeners y notificar los cambios, al igual que una interfaz donde se define el método que es implementado en el controlador de nuestro chat. El *ControladorChat* implementa *MensajesListener*. Esta jerarquía le permite tener la funcionalidad *nuevosMensajes*, que es la función que recibirá el evento con la lista de *MensajeWhatsApp* y los transformará a nuestros mensajes.
  Para cargar estos mensjaes utilizamos el *CargadorMensajes*, a cuyo método *setMensajes* se llamará desde el controlador, tras ser seleccionado en la ventana principal indicando la ruta del fichero donde está el chat y la plataforma. Esto activará todos los *Listeners*, lo que a su vez activará la función *nuevosMensajes* del controlador que hemos mencionado anteriormente.

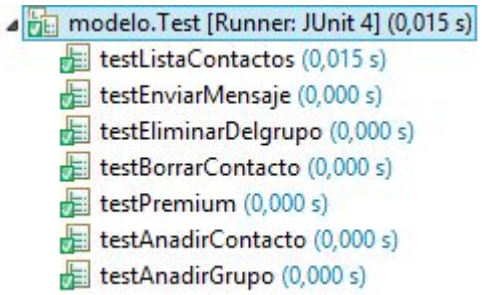
6. TEST UNITARIOS IMPLEMENTADOS

A continuación mostramos los test unitarios implementados para comprobar el correcto funcionamiento de algunas de las funciones principales, concretamente hemos comprobado la correcta creación de un contacto y que se guarde correctamente en la lista de contactos,, la creación de un grupo por parte de un usuario y que dicho grupo se añade a la lista de grupos administrados por el usuario, la correcta conversión de un usuario a premium, eliminar a un contacto de un grupo y comprobar que efectivamente dicho contacto ya no pertenece al grupo, enviar un mensaje y que se añada correctamente a la lista de mensajes del receptor y por último la eliminación de un contacto de las lista de contactos de un usuario.

```
Usuario usuario;
Usuario usuario1;
ContactoIndividual contacto;

@Before
public void crearUsuario() {
    usuario = new Usuario("Pepe", Date.valueOf(LocalDate.of(1998, 4, 2)), "696140537", "pepeg", "11");
    usuario1 = new Usuario("Juan", Date.valueOf(LocalDate.of(1998, 4, 2)), "602548792", "juanpe", "11")
    contacto = new ContactoIndividual("Juan", "602548792", usuario1);
}


@org.junit.Test
public void testAnadirContacto() {
    ContactoIndividual c = usuario.addContacto("Juan", "602548792", usuario1);
    assertEquals(c.getMovil(), contacto.getMovil());

}
@org.junit.Test
public void testListaContactos() {
    usuario.addContacto(contacto);
    assertEquals(usuario.getContactos().size(), 1);

}


@org.junit.Test
public void testAnadirGrupo() {
    LinkedList<ContactoIndividual> miembros = new LinkedList<ContactoIndividual>();
    miembros.add(contacto);
    usuario.addGrupo("Grupo", miembros);
    assertEquals(usuario.getGruposAdmin().size(), 1);

}
```

```
@org.junit.Test
public void testPremium() {
    usuario.setPremium(true);
    assertTrue(usuario.isPremium());
}

@org.junit.Test
public void testEliminarDelgrupo() {
    LinkedList<ContactoIndividual> miembros = new LinkedList<ContactoIndividual>();
    miembros.add(contacto);
    usuario.addGrupo("Grupo", miembros);
    Grupo g = usuario.getGruposAdmin().get(0);
    g.removeContacto(contacto);
    assertEquals(g.getContactos().size(), 0);

}
@org.junit.Test
public void testEnviarMensaje() {
    usuario.enviarMensajeEmisor("hola", LocalDateTime.of(2018, 10, 10, 11, 25), 0, usuario, contacto);
    assertEquals(contacto.getMensajes().size(), 1);

}

@org.junit.Test
public void testBorrarContacto() {
    usuario.addContacto(contacto);
    usuario.removeContacto(contacto);
    assertTrue(usuario.getContactos().isEmpty());

}
```

Finalmente mostramos una captura con el resultado de los test donde se puede ver que todas las pruebas han resultado exitosas.



7. MANUAL DE USUARIO

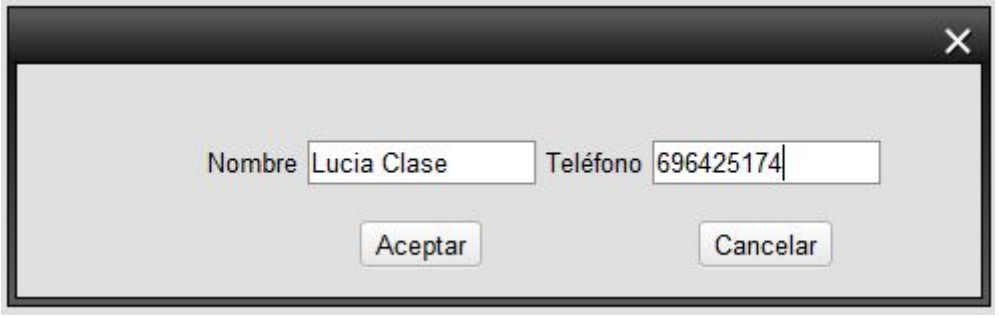
Cuando arrancamos la aplicación nos encontramos con una ventana de inicio, la cual nos pedirá nuestro usuario y contraseña. Si no tenemos cuenta nos dará la posibilidad de registrarnos haciendo click en Registro.

Para registrarnos debemos rellenar nuestros datos. La propia aplicación verificará si el teléfono y el nombre de usuario son válidos.

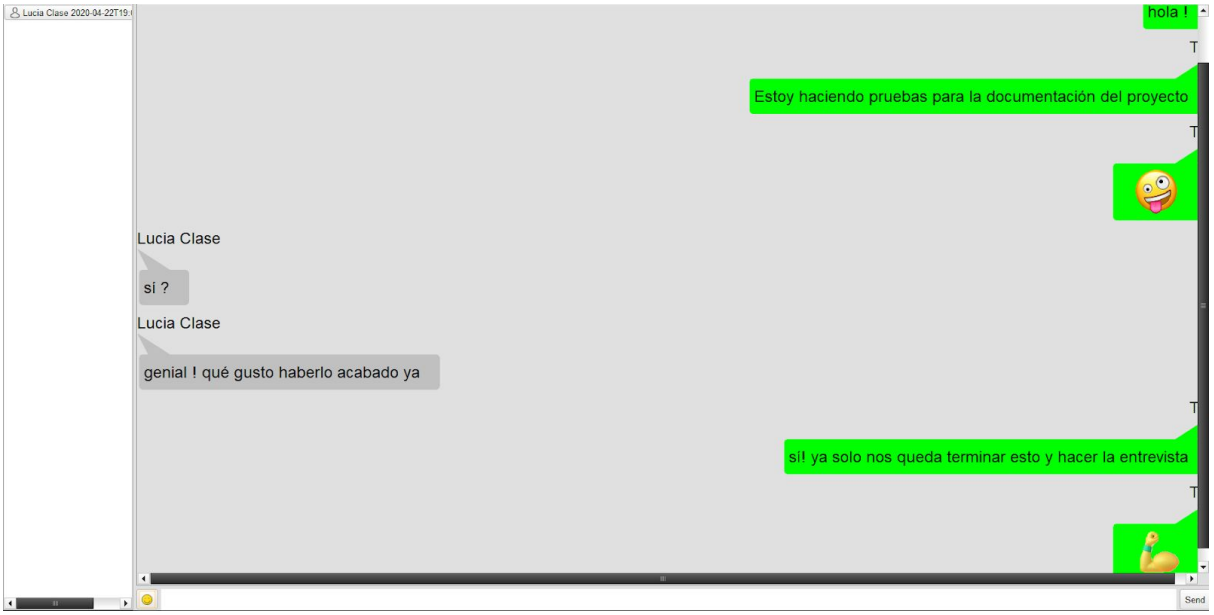
Una vez dados de alta, entramos a la ventana principal de la aplicación en la que se mostrará (de izquierda a derecha) la posibilidad de filtrar mensajes, nuestro usuario, un menú con opciones, la posibilidad de importar una conversación de Whatsapp y la posibilidad de borrar una conversación.

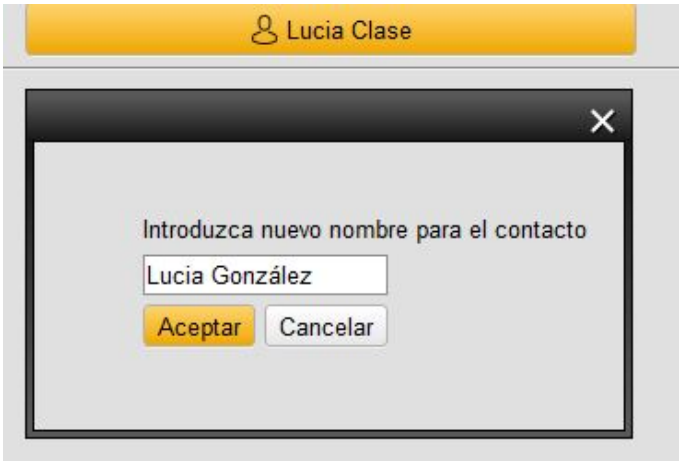
Este es el menú de opciones.

A continuación, vamos a intentar añadir a un contacto (Este contacto debe estar registrado como usuario para que podamos encontrar su número de teléfono).



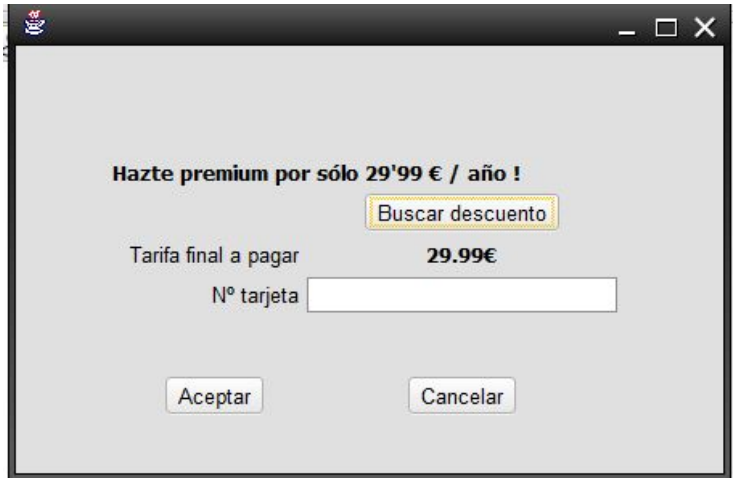
Ahora podemos mantener una conversación perfectamente con nuestro contacto:





Pulsando en el contacto, podemos cambiarle el nombre, que lo actualizará automáticamente en la lista de la izquierda.

Como hemos señalado anteriormente, con el menú de opciones se pueden realizar múltiples acciones como mostrar los contactos, que nos desplegará una tabla con el nombre del contacto, el teléfono, la foto de perfil y los grupos que tenemos en común.
Estos grupos se crean con la opción crear grupo, que nos pedirá un nombre para el grupo y nos desplegará nuestra lista de contactos para añadirlos. Si lo que queremos es modificar un grupo existente, con la opción modificar grupo tendremos que seleccionar el grupo que queremos editar y posteriormente se nos desplegará este con sus miembros y nuestra lista de contactos para añadir a nuevos.



Tenemos también la opción de convertirse en usuario premium, lo que nos dará acceso a las siguientes dos funcionalidades. Para convertirse en premium sólo hay que introducir el número de la tarjeta y aceptar. Es posible buscar tus descuentos en función del número de mensajes o el tiempo que lleva registrado el usuario.

Las opciones del premium son poder generar las estadísticas de los mensajes enviados cada mes y generar un PDF.

Finalmente, podemos cerrar la sesión de un usuario.

El último botón de la derecha permite borrar todos los mensajes en una conversación con un contacto. Tras pulsarlo, todo el panel del chat aparecerá vacío.

8. OBSERVACIONES FINALES

Una vez acabado el proyecto, podemos afirmar sin lugar a dudas que ha sido la práctica más completa de lo que llevamos de grado. No sólo se han puesto a prueba nuestra capacidad de programar en Java, sino que también hemos tenido que demostrar la capacidad de trabajar con interfaces GUI, la herramienta de Maven, Git...

Estimamos haber invertido en el proyecto entre los dos componentes del grupo unas 150 horas de trabajo. Mucha parte del tiempo se nos fue al principio creando las ventanas, ya que es la primera vez que trabajábamos con Swing y según avanzábamos en la parte del modelo y la persistencia nos dábamos cuenta de que debíamos ir haciendo cambios en estas.

La mayor dificultad la encontramos en incorporar la persistencia a la aplicación, ya que cualquier mínimo error en esta causaba errores que las primeras semanas no sabíamos ni cómo detectar.

Queremos destacar también al profesor, por habernos atendido incluso en horario fuera de tutorías cuando estábamos perdidos, de lo contrario no habríamos sabido cómo seguir en ciertas ocasiones y no nos habría sido posible acabar el proyecto.