Trabajo Práctico 1 — Smalltalk

[7507/9502] Algoritmos y Programación III
Curso 2
Primer cuatrimestre de 2018

Alumno:	COLOMBO, Juan Ignacio
Número de padrón:	103471
Email:	juanicolombo@icloud.com

Índice

1. Introducción

El presente informe reune la documentación de la solución del primer trabajo práctico de la materia Algoritmos y Programación III que consiste en desarrollar una aplicación de estilo Whatsapp en Pharo utilizando los conceptos del paradigma de la orientación a objetos vistos hasta ahora en el curso.

2. Supuestos

En la implementacion del tp, no es correcto enviar un mensaje a un usuario que no se halla agregado previamente, ya que este tiraria un error, al no poder ser encontrada la clave en el diccionario, lo mismo pasaria con los canales y conversaciones, con lo cual se debe siempre agregar ya sea un usuario, o canal al AlgoChat antes de poder realizar operaciones sobre los mismos.

3. Modelo de dominio

El AlgoChat es la clase que recibe y maneja todo el programa. Esta misma recibe los mensajes, como asi almacena los usuarios, canales y conversaciones que se vayan creando. El diseño consiste principalmente, en tres diccionarios, uno para los usuarios, otro los canales y otro las conversaciones. Cuando se quiere accceder a alguno de estos, se ahce mediante el nombre del canal , usuario o lo que quiera, y este devolvera una clase, para asi poder realizar las operacionez que se quiera. El AlgoChat tiene como funciones principales la inicializacion de las clases usuario, canal o conversaciones, a medida que se vaya agregando una de estas al chat. Luego tambien distribuye los mensajes que se reciben entre las clases, ya sea enviarle un mensaje a un usuario en particular, a alguna conversacion, la cual cuenta con usuarios dentro, y estos deberan recibir el mismo mensaje tambien, y por ultimo los mensajes a los canales, siendo estos los mas complejos, ya que el echo de que un usuario este en el canal no significa que vaya a recibir el mensaje. Finalmente el AlgoChat se encarga tambien de el manejo de la devoluvion de mensajes que posea cada clase.

4. Diagramas de clase

En la Figura 1 podemos ver la implementacion general de las clases. Podemos ver claramente como el AlgoChat, tiene como funciones principales, la de agregar un nuevo canal, usuario o conversaciones, asi como la distribucion de los mensajes. Luego vemos como los canales, tiene como funcion principal la de agregar usuarios o devolver los mensajes que se le envian. En el caso de conversaciones lo mismo. En el caso de Usuario, vemos que es la clase que mas se conecta con las demas, esto se debe a que un AlgoChat puede tener muchos usuarios, como a la ves un canal o conversacion.

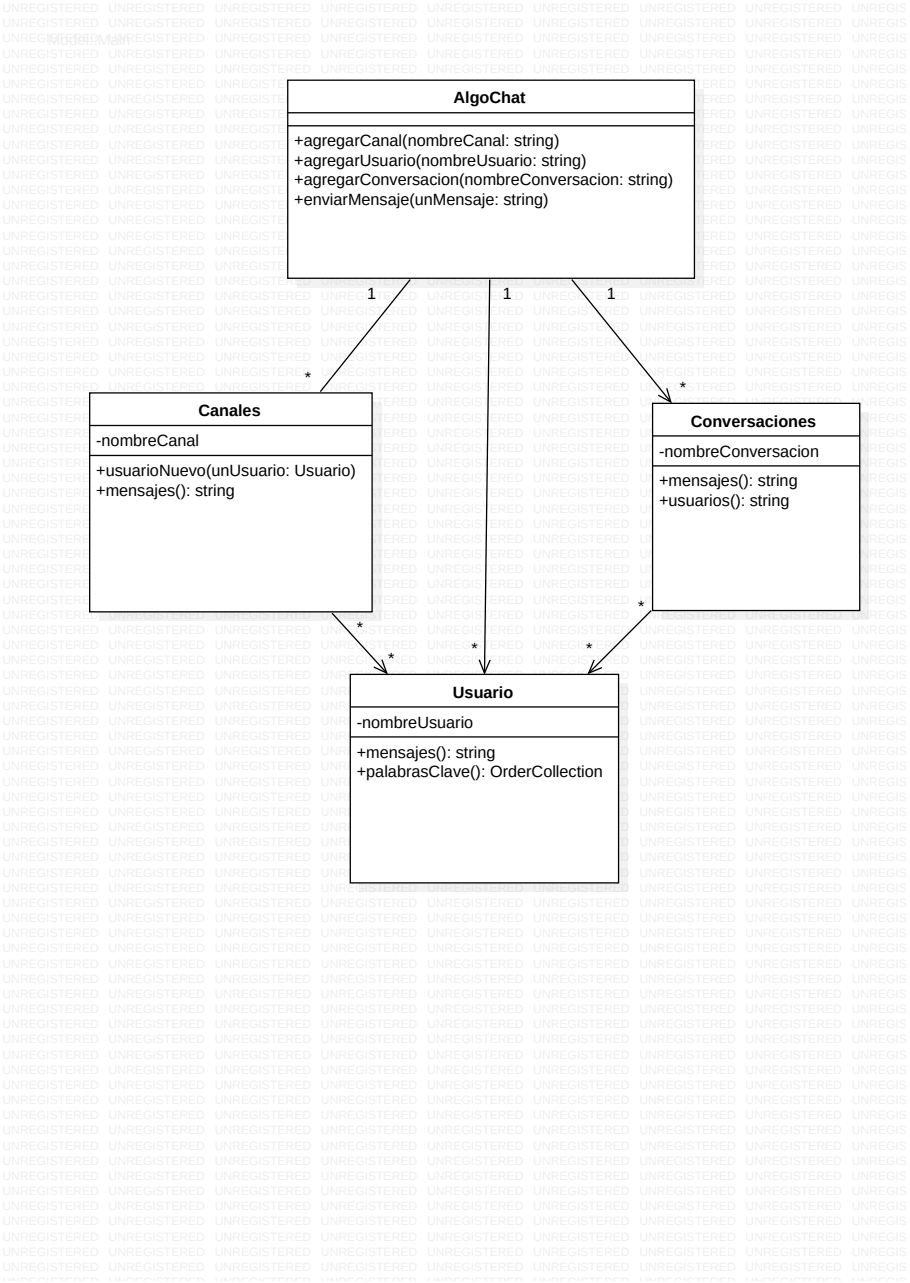

Figura 1: Diagrama de distribucion de mensajes.

En la Figura 2, podemos observar como se comportan los mensajes, que seria la idea fundamental del AlgoChat. Podemos ver que los canales tienen usuarios, a los que se pueden agregar al canal, y enviar mensajes.Los usuarios, a su ves como los canales, cuentan con notificaciones, que es donde se almacenan los mensajes que recibe cada uno, para luego devolverlos en el formato que se desee, ya sea normal, o acortado a una longitud que se quiera.

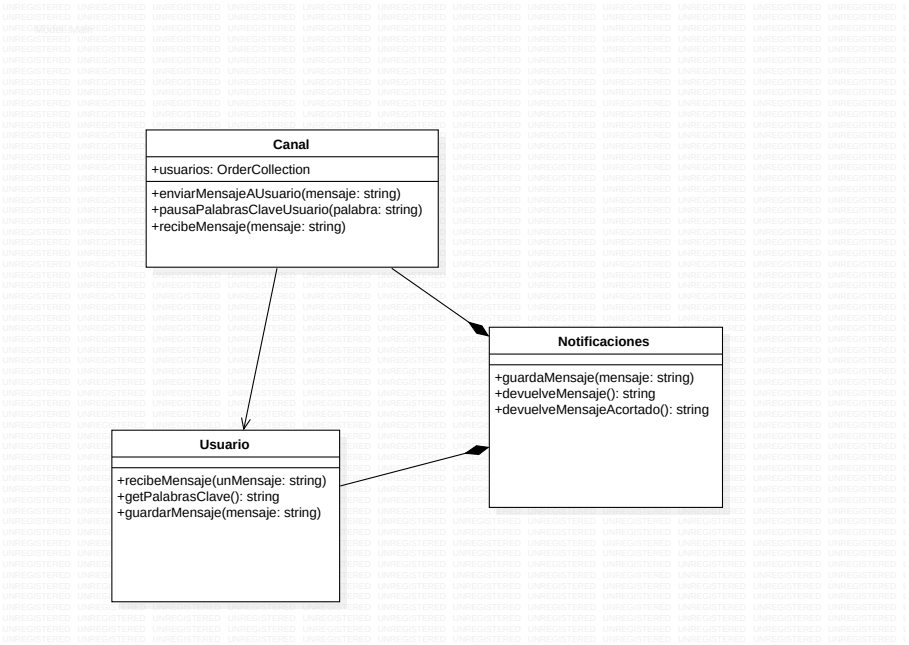

Figura 2: Diagrama del AlgoChat.

5. Detalles de implementación

5.1. Implementacion de la clase Canal

La clase canal es en si la mas compleja del trabajo. Esto se debe al echo de que los usuarios que estan en el canal , no reciben si o si el mensaje, esto solo pasa si son mencionados o el mensaje que recibe el canal contiene una palabra clave del usuario. Es asi que cuando se quiere publicar un mensaje en el canal, se debe ver a quien enviar este mensaje de los usuarios dentro. Esto se hace gracias a que el canal cuenta con una lista de usuarios, a los que al momento de distribuir el mensaje se va a recorrer en busca de si cumplen los criterios o no. Esto sucede en el metodo de comprobarPalabrasClave: deCanal:.

```
  mensajeALista := (mensajeRecibido splitOn: ' ').

listaDeUsuarios := self getUsuarios.

listaDeUsuarios do: [ :unUsuario |

listaPalabras := (unUsuario getPalabrasClaveDeCanal: unCanal).

(listaPalabras includesAny: mensajeALista ) ifTrue:[ unUsuario guardarNotificacion: mensajeRecibi
]
 ]
```

Aca podemos ver el codigo de el metodo mencionado. En este se agarra el mensaje a analizar y se lo splitea, generando una lista en donde los elementos son cada palabra del mensaje. Luego se agarra la lista de usuarios que hay en el canal, para asi poder recorrerla. Como dijimos antes cada usuario tiene sus palabras claves y su nombre respectivo, para ver si reciben el mensaje o no. Entonces lo que se hace el obtener una lista de las palabras clave por usuario, y se fija si

algun elemento de esta concuerda con la lista de palabras del mensaje recibido. En caso que alguna palabra coincida, el mensaje es enviado a los usuarios, para que lo almacenen en sus notificaciones.

5.2. Uso de la clase Notificaciones

La clase notificaciones, es una clase de uso comun. Esta se inicializa cuando se crea una clase de usuario, canal o conversacion. Su rol es el de almacenar los mensajes en una lista, para luego al momento de que alguna de las otras clases quiera ver sus mensajes, simplemente se lo pida a Notificaciones. Esta clase es individual para cada usuario, cada canal y conversacion, debido a que cada uno cuenta con distintas notificaciones. Las funciones principales son entonces la de almacenar los mensajes , y devolverlos ya sea de forma normal, la cual sera uniendo la lista mediante el string ' | '. O tambien se puede querer devolver el mensaje acortado a un largo dabo.

```
| listaNotificacionesResumida |


listaNotificacionesResumida := notificacion  collect: [ :mensajeViejo| (mensajeViejo contractTo:

^(listaNotificacionesResumida joinUsing: ' | ').


 ]
```

Aca podemos ver el codigo para resumir la lista de mensajes. Lo que hace es contraer cada elemento de la lista a un largo recibido , para luego juntar la lista, como se hace con la de los mensajes normales y devolverlo.

6. Diagramas de secuencia

En la Figura 3, podemos ver el diagrama de secuencia que muestra, como se publica un mensaje en el canal. Podemos ver como algochat recibe un mensaje, el cual es enviado a el canal y una ves aca realiza dos acciones. Primero guarda el mensaje en las notificaciones del canal.Luego se pasa a comprabar a que usuarios dentro del canal hay q mandarle el mensaje. Esto se hace convirtiendo el mensaje a lista y obteniendo los usuarios en el canal. Podemos ver que una ves que se obtiene la lista de usuarios esta se recorre en un ciclo, para cada uno de ellos. Se le pide a cada usuario sus palabras clave, y se fija si alguna palabra clave concuerda con la lista de palabras del mensaje. En el caso de que no halla coincidencia, no pasa nada, de lo contrario el mensaje es enviado al usuario, el cual lo envia a las notificaciones para ser almacenado en los mensajes de usuario.

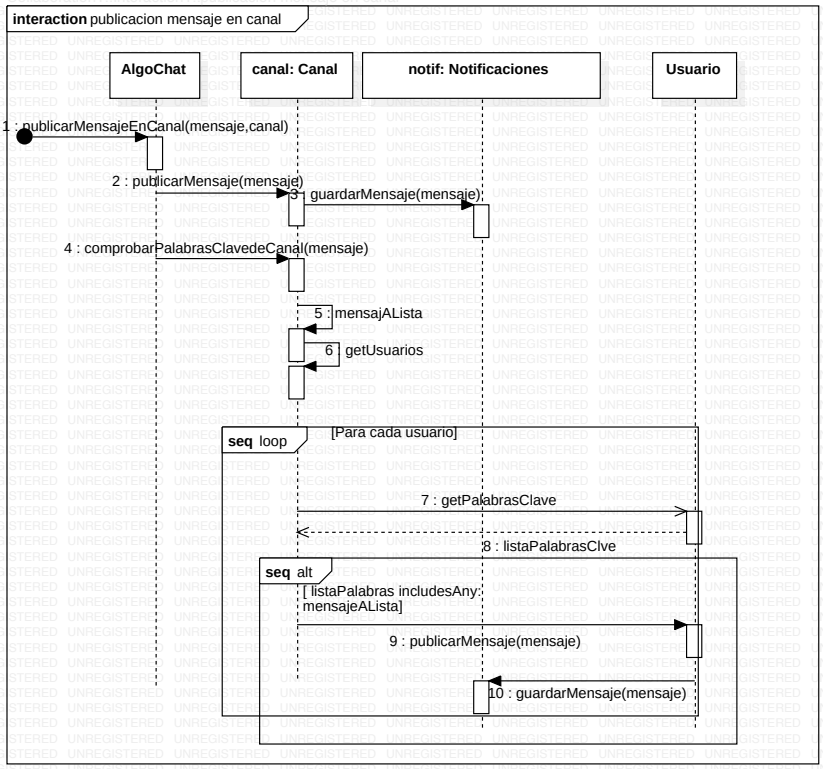

Figura 3: Publicacion Mensaje en canal

En la figura 4, tenemos otro diagrama, el cual representa como funciona, el AlgoChat cuando se le pide que devuelva las notificaciones, en este caso para un usuario, pero seria para los canales y notificaciones, el mismo prosedimiento, solo que se le manda el mendaje de getNotificaciones, a la clase que se quiera obtener.En este caso tambien vemos como se pide las notificaciones resumidas, cosa que seria el mismo proceso si las quisieramos normales, cambiando los nombres de las llamadas. Entonces en el diagrama vemos como se le pide a AlgoChat las notificaciones para un usuario dado, y un largo pasado, y el algo chat le manda esto al usuario, con el largo recibido. El usuario para a pedirle a notificaciones que le devuelva los mensajes previamente guardados ahi, achicados cada una al largo recibido, y la clase notificaciones, devuelve los mensajes al usuario, el cual se los devuelve a AlgoChat.

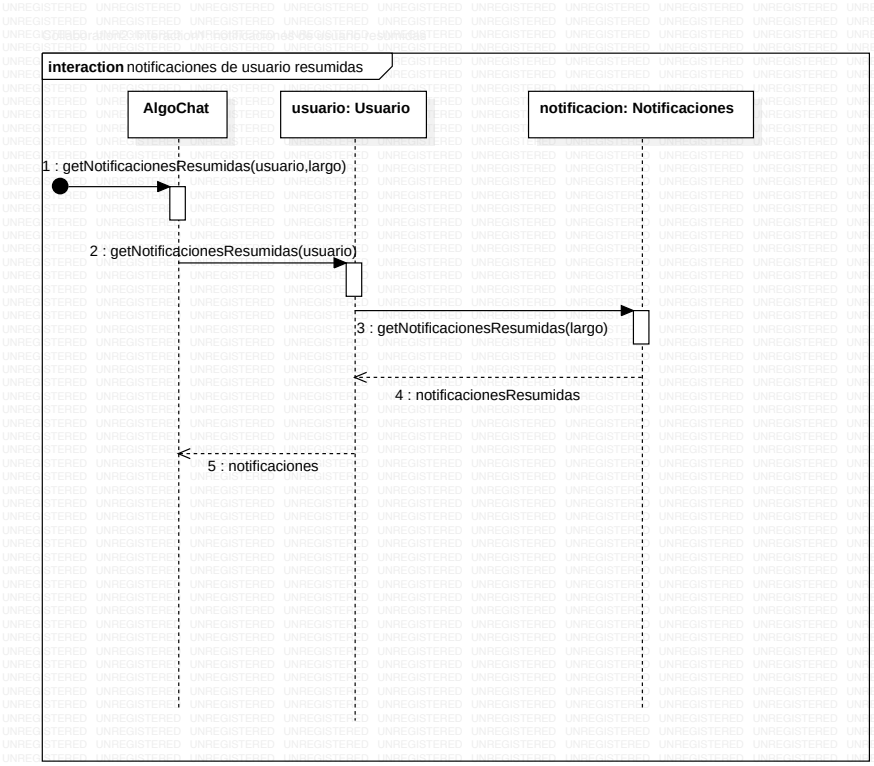

Figura 4: Devolucion de MensajesResumidos

En la Figura 5, se muestra como funciona una nueva conversacion.Cuando a AlgoChat se le pide que haga una nueva conversacion, se le pasa el nombre de la misma, el cual son los usuarios separados por coma. El mismo AlgoChat splitea el string, generando una lista de usuarios. Una ves obtenida esta lista realiza un ciclo recorriendo cada usuario, y pasandoselo a la conversacion, para que lo agrega a la lista de usuarios que posee la clase conversacion.

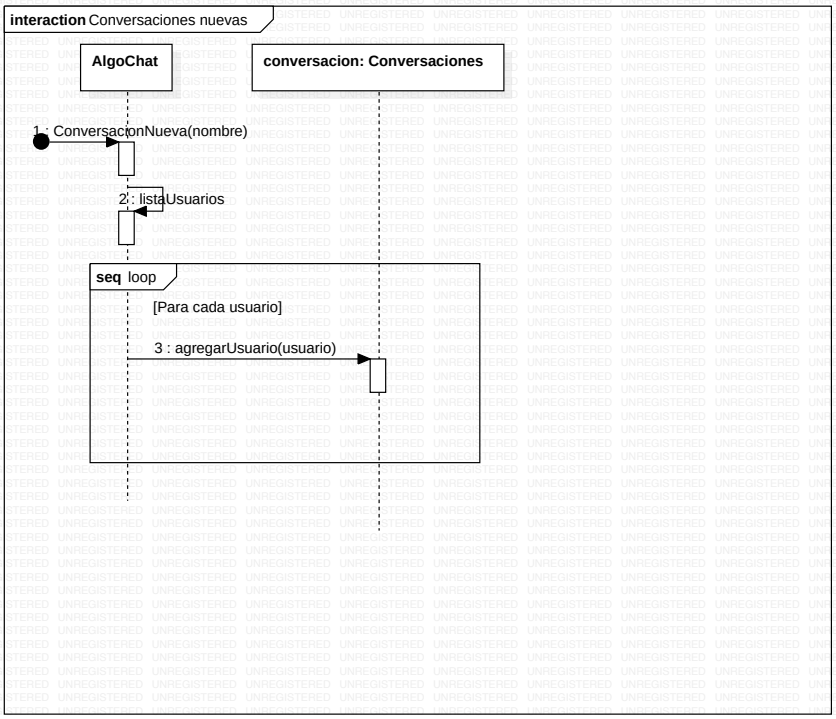

Figura 5: Comportamiento nueva conversacion