Manual de instalación y puesta en marcha

Para ejecutar la práctica es necesario tener instalado MYSQL. La base de datos llamada ProyectoTwitter viene disponible en el directorio databaseFiles. Se trata de un archivo SQL que puede ser importado desde MYSQL para crear las tablas necesarias.

Para la configuración de los parámetros de conexión a la base de datos hay que rellenar el fichero dbconfig.cnf indicando en la primera linea el nombre de la base de datos, en la segunda el usuario con el que conectar, en la tercera la contraseña y en la cuarta (opcional) el número de puerto en el que está corriendoel servidor de la base de datos.

Tambien es necesario tener la version 1.7 de java ya que la interfaz gráfica está hecha en JavaFX.
Junto con la práctica viene incluido un script con nombre ejecuta.sh. Este script pondrá en ejecución el rmiregistry, el servidor de nuestro twitter y un cliente con el cual se podrá acceder tanto a nuestro twitter como al twitter real con solo elegir la opción correspondiente en el login.

Aclaraciones
Al conectar con el twitter real el cliente se vuelve significativamente más lento al tener que realizar constantemente peticiones de datos de tweets y usuarios al servidor a través de internet. JavaFX no cargará la interfaz hasta tener toda la información necesaria. Por ejemplo, al iniciar el cliente se debe cargar el timeline del usuario, con los últimos tweets recibidos por el usuario. Este proceso suele ser largo pero una vez que dispone de la información necesaria, el cliente cargará la interfaz idéntica a la ofrecida en nuestro twitter basado en RMI. Cualquier función del cliente será significativamente más lenta si se realiza en con el twitter real.
También hay que tener en cuenta que twitter no permite más de 350 consultas a su servidor en una hora a través del api y esto suele dar problemas cuando se realizan búsquedas, ya que requieren muchas consultas. Twitter recomienda el uso de su Streaming API para evitar el problema del limite de peticiones y acceder con baja latencia a los datos de su servidor.

Análisis de JTwitter

Jtwitter se trata de una librería desarrollada por Winterwell que permite el acceso y consulta a la API de Twitter. Tiene un tamaño reducido en comparación con otras librerías para el mismo cometido y prácticamente permite realizar las mismas acciones que se pueden realizar desde la web de Twitter. Su código fuente está disponible desde la web de Winterwell.

Está formada por 4 paquetes:

- winterwell.jtwitter
- winterwell.jtwitter.android
- winterwell.jtwitter.ecosystem
- winterwell.jtwitter.guts

Para el análisis nos vamos a centrar en el paquete “winterwell.jtwitter" ya que el resto de paquetes no nos van a ser útiles ya que simplemente contienen algunas utilidades (conversión a base64, conexión desde android...) que no necesitamos explícitamente.

El paquete de jtwitter contiene todas las clases, interfaces, enum y excepciones necerias para realizar todas las acciones con la API de Twitter. A continuación pasamos a describirlas marcando en negrita las más importantes.

Interfaces

- **Astream.IListen**: se trata de una interfaz que todas las clases que quieran escuchar eventos (nuevos tweets, eventos follow, cambios de propiedades de usuario, favoritos...) deben implementar. Está formada por tres métodos:
  - `boolean processEvent(TwitterEvent event): método para notificar eventos. Más adelante se detallan los diferentes eventos.`
  - `boolean processSystemEvent(Object[] obj): para notificar eventos del sistema, como límites...`
  - `boolean processTweet(Twitter.ITweet tweet): para procesar tanto nuevos tweets como mensajes privados.`

- **Twitter.ICallback:** se utiliza para el callback de busquedas que pueden requerir "bastante" tiempo.

- **Twitter.IHttpClient:** interfaz para clases que actúan como cliente HTTP.

- **Twitter.ITweet**: es una interfaz que se utiliza tanto para los Status (tweets) como para los Message (mensajes privados). Está formada por los siguientes métodos:
  - `Date getCreatedAt(): fecha en la que fue creado.`
  - `String getDisplayText(): texto tal y como se va a mostrar, con las urls reemplazadas.`
  - `Number getId()`
  - `String getLocation()`
  - `List<String> getMentions(): lista de "screenNames" que hay en este mensaje.`
  - `Place getPlace(): localización del mensaje.`
  - `String getText(): texto del mensaje.`
  - `List<Twitter.TweetEntity> getTweetEntities(Twitter.KEntityType type): lista de entities (pueden ser urls, menciones o hashtags).`
  - `User getUser(): usuario que creó el mensaje.`

- **TwitterEvent.Type:** contiene los diferentes tipos de TwitterEvent. Son los siguientes:
  - `static String ADDED_TO_LIST`
  - `static String FAVORITE`
  - `static String FOLLOW`
  - `static String LIST_CREATED`
  - `static String REMOVED_FROM_LIST`
  - `static String UNFAVORITE`
  - `static String USER_UPDATE`

Clases

- **AStream:** Dentro de Jtwitter esta sería la interfaz que se encarga del callback, por decirlo de algún modo. Consta de dos threads, el primero es el que se encarga de procesar todo y luego otro que se encarga de acceder a este mediante polling.

- **AStream.Outage:** Clase que sirve para registrar los datos de un outage de la red.

- **InternalUtils:** clase con ciertas utilidades.

- **Message:** clase que representa un mensaje privado. Implementa la interfaz de Twitter.ITweet y le añade los métodos equals, hashCode y toString.

- **OAuthSignposClient:** Esta clase implementa el protocolo Oauth que establece un método seguro de autenticación en una aplicación tipo cliente (como nuestro cliente de twitter) que requiere acceso a una cuenta en un servidor. En el caso de twitter un usuario de nuestro cliente nunca escribe su contraseña en nuestro cliente, solo registra su nombre y mediante Oauth se le redirige a un formulario (en twitter.com) para autenticarse. Al introducir los datos correctamente twitter genera un ping aleatorio el cual debe introducir en el cliente para de esta forma autorizar la conexion de nuestro cliente con el servidor de twitter.

- **Place:** Un place simboliza un lugar. Se utiliza para la geolocalización (Véase Twitter_Geo).

- **RateLimit:** para manejar los límites de uso (Twitter tiene una limitación de peticiones a su API, si esta se sobrepasa puede acaba añadiéndote a la lista negra).

- **Status:** clase que representa un tweet. Implementa la interfaz de Twitter.ITweet. Además de los métodos que le añadeMessage, también tiene otros métodos como isFavorite (para saber si es un tweet favorito del usuario actual) e isSensible (para ver si tiene contenido no apto para todos los públicos). En sus variables públicas almacena información como el id del cual es respuesta este tweet o el número de retweets que han hecho de este tweet.

- **Twitter:** es la clase principal que hace uso y devuelve objetos de las demás. COntiene una gran cantidad de métodos, muchos de ellos "deprecated" debido a ser extraidos en otros clases, cambios en sus nombres, etc. Entre los métodos no *deprecated* podemos destacar:
  - `Twitter_Account account(): devuelve un objeto con el que acceder a métodos de la API relacionados con las cuentas de usuario.`
  - `static int countCharacters(String statusText): obtiene el tamaño efectivo del texto, teniendo en cuenta que Twitter cuenta las URLs como 20 caracteres, independientemente de la longitud que tengan.`
  - `void destroy(Twitter.ITweet tweet): borra el Status o Message indicado.`
  - `void destroyMessage(Number id) : borra el mensaje privado con el ID indicado.`
  - `void destroyStatus(Number id): borra el tweet indicado.`
  - `List<Message> getDirectMessages(): obtiene los mensajes privados del usuario actual.`
  - `List<Message> getDirectMessagesSent(): obtiene los mensajes privados enviados por el usuario actual.`
  - `List<Status> getFavorites(): obtiene los tweets marcados por el usuario actual como favoritos.`
  - `List<Status> getFavorites(String screenName): obtiene los tweets favoritos del usuario indicado.`
  - `List<Status> getHomeTimeline(): obtiene el timeline del usuario actual; es decir, los tweets más recientes del usuario actual y los usuarios a los que este sigue.`
  - `List<Status> getMentions(): devuelve los tweets en los que el usuario actual ha sido mencionado.`
  - `List<User> getRetweeters(Status tweet): obtiene los usuarios que han retweeteado el tweet dado.`
  - `List<Status> getRetweets(Status tweet)`

- List<Status> **getRetweetsByMe**(): obtiene los tweets que el usuario actual ha retweeteado.
- List<Status> **getRetweetsOfMe**(): obtiene los tweets del usuario actual que otros usuarios han retweeteado.
- String **getScreenName**()
- User **getSelf**(): devuelve un objeto User con el que poder acceder a métodos para realizar acciones sobre el usuario.
- Status **getStatus**(): devuelve el último tweet del usuario actual.
- Status **getStatus**(Number id): devuelve el tweet con el ID dado.
- Status **getStatus**(String username): devuelve el último tweet del usuario indicado con su screenName.
- List<Status> **getUserTimeline**(): obtiene los tweets más recientes del usuario actual.
- List<Status> **getUserTimeline**(Long userId): obtiene los tweets más recientes del usuario con ese ID.
- List<Status> **getUserTimeline**(String screenName): obtiene los tweets más recientes del usuario con ese screenName.
- Status **retweet**(Status tweet): hace un retweet de un tweet.
- List<Status> **search**(Strings earchTerm): busca un término en Twitter.
- List<Status> **search**(String searchTerm, Twitter.ICallback callback, int rpp): busca un término en Twitter.
- Message **sendMessage**(String recipient, String text): envía un mensaje (text) privado al usuario indicado con su screenName (recipient) desde el usuario actual.
- Status **updateStatus**(String statusText): publica un nuevo tweet.
- Status **updateStatus**(String statusText, Number inReplyToStatusId): publica un nuevo tweet en respuesta a otro.
- Twitter_Users **users**(): obtiene un objeto para realizar acciones relacionados con los usuarios de Twitter.

- **Twitter_Account:** se trata de una clase con la que se puedne realizar algunos de las acciones relacionadas con los usuarios de Twitter. Contiene algunos métodos para crear y guardar búsquedas. Destacamos sobretodo el método

  User **setProfile**(String name, String url, String location, String description)
  con el que se pueden actualizar ciertos valores del perfil del usuario.

- **Twitter_Account.Search:** clase que representa una búsqueda guardada.

- **Twitter_Analytics:** por ahora solo permite saber cuántas veces ha sido tweeteada una url.

- **Twitter_Geo:** clase que se utiliza para acciones relacionadas con la geolocalización. Requiere el paquete winterwell.jgeoplanet. Contiene métodos para obtener las coordenadas de la localización y su nombre, devolviéndolas en forma de Place. Se utiliza también para búsquedas de usuarios dentro de un lugar.

- **Twitter_Users:** se trata de una clase que provee métodos con los que realizar acciones sobre los usuarios y las relaciones entre ellos. Entre los métodos que contiene esta clase podemos destacar:
  - `User follow(String username): para empezar a seguir al usuario indicado.`
  - `User follow(User user): para empezar a seguir al usuario indicado.`
  - `List< Number> getFollowerIDs(): devuelve una lista de usuarios que están siguiendo al usuario actual.`
  - `List< Number> getFollowerIDs(String screenName): devuelve una lista de usuarios que están siguiendo al usuario indicado.`
  -
  - `List< Number> getFriendIDs(): devuelve una lista de usuarios a los que está siguiendo el usuario actual.`
  - `List< Number> getFriendIDs( String screenName): devuelve una lista de usuarios a los que está siguiendo el usuario indicado.`
  - `User getUser(long userId): obtiene una instancia de esta clase para un usuario dado.`
  - `User getUser( String screenName): obtiene una instancia de esta clase para un usuario dado.`
  - `isFollower( String followerScreenName,  String followedScreenName): indica si un usuario dado está siguiendo a otro usuario dado.`
  - `boolean isFollowing( String userB): indica si el usuario actual está siguiendo  a un usuario dado.`
  - `boolean isFollowing(User user): indica si el usuario actual está siguiendo  a un usuario dado.`
  - `List<User> searchUsers( String searchTerm): busca usuarios que contengan el término buscado en su nombre.`
  - `User stopFollowing( String username): deja de seguir a un usuario dado.`
  - `User stopFollowing(User user) : deja de seguir a un usuario dado.`
  - `boolean userExists( String screenName): indica si el usuario indicado existe.`

- **Twitter.TweetEntity:** clase que representa una entity de Twitter.

- **TwitterEvent:** clase que representa un evento. Destacamos los siguientes métodos:
  - `Date getCreatedAt(): fecha en la que se produjo el evento.`
  - `User getSource(): usuario que originó el evento.`
  - `User getTarget(): usuario que se vio afectado por el evento.`
  - `Object getTargetObject(): objeto afectado si no se trataba de un usuario.`
  - `String getType(): devuelve el tipo de evento (los de la interfaz TwitterEvent.Type)`

- **TwitterList** : lista de usuario que hace carga bajo demanda de sus entradas.

- **TwitterStream:** clase para la conexión con la API stream.

- **URLConnectionHttpClient:** un cliente HTTP.

- **User:** clase que representa a un usuario y las acciones que se pueden realizar sobre él, sobretodo métodos *getter* para obtener información del usuario. Entre sus métodos destacamos los siguientes:
  - `Date getCreatedAt(): fecha en que se registró el usuario.`

    - String **getDescription**(): descripción del usuario.
    - int **getFavoritesCount**(): número de favoritos del usuario.
    - int **getFollowersCount**() : número de seguidores del usuario.
    - int **getFriendsCount**() : número de usuarios a los que sigue este usuario.
    - Long **getId**()
    - String **getName**(): nombre real del usuario.
    - Place **getPlace**()
    - URI **getProfileImageUrl**(): devuelve la URL de la imagen de perfil.
    - boolean **getProtectedUser**()
    - String **getScreenName**(): apodo del usuario en Twitter.
    - Status **getStatus**(): último tweet del usuario.
    - int **getStatusesCount**(): número de tweets del usuario.
    - Boolean **isFollowedByYou**(): si el usuario actual está siguiendo a este usuario.
    - Boolean **isFollowingYou**(): si este usuario está siguiendo al usuario actual.
    - boolean **isProtectedUser**(): si se trata de un usuario con protección (privado).
- UserStream: clase *deprecated* para la conexión con la API stream.

Conclusiones

Una vez realizado el análisis de la librería pudimos darnos cuenta de que la interfaz no era clara, no estaba correctamente planificada y parecía que se había ido desarrollando en base a las necesidades que iban surgiendo.

Hay ciertas cosas que carecen de sentido, como por ejemplo que los valores de las propiedades del perfil de un usuario se guarden desde Twitter_Acount con setProfile pero se obtengan desde User...deberían estar en la misma clase.
Tampoco estamos de acuerdo en que las clases tengan bastantes atributos públicos en vez de métodos con los que obtener el valor.

En nuestra opinión, la librería todavía no es lo suficientemente buena para implementar su interfaz en nuestro servidor. Sin embargo, dado que teníamos que realizar un cliente que fuese capaz de conectarse tanto a nuestro servidor de Twitter como al servidor real (mediante JTwitter) nos vimos obligados a implementar parte de la interfaz en nuestro servidor para no tener que hacer dos códigos distintos para las dos conexiones.

Durante el desarrollo del proyecto, hemos tenido que modificar el código fuente de JTwitter para añadirle cosas que le faltaban (ciertos tipos de evento, algunos métodos, etc). Además de tener que generar un conjunto de interfaces similares a las clases descritas anteriormente y tener que modificar los nombres de las clases y hacer que implementen las interfaces con su nombre (por ejemplo, la clase Twitter pasa a llamarse TwitterImpl y TwitterImpl implementa la interfaz Twitter).

Otro problema importante que tiene esta libreria es que al no estar optimizada y requiere demasiadas consultas al servidor de twitter. Esto no seria un problema si no fuera porque twitter solo permite 350 peticiones por hora para un cliente. En la web recomiendan usar twitterStream, una librería que evita este tipo de problemas. Esto afecta el funcionamiento de nuestro cliente y es normal que luego de estar un rato usándolo se pierda la conexión con el servidor de twitter, lanzando una excepción RATELIMT que no es más que lo explicado anteriormente.

Breve descripción del sistema

En general, el sistema se puede dividir en 3 partes principales: el cliente, el servidor y un conjunto de interfaces comunes. Todo el código del cliente (exceptuando la parte de login y conexión) utiliza esas interfaces comunes. Tanto nuestro servidor como nuestro paquete modificado de JTwitter implementan esas interfaces; de esta manera, con el mismo código se puede conectar a dos elementos diferentes como son nuestro servidor y la librería de JTwitter.

Nuestro servidor de Twitter utiliza la tecnología Java RMI para proporcionar al acceso a unos métodos dados por las interfaces comunes. Durante la fase de conexión en el cliente, se obtiene un objeto que implementa la interfaz Twitter gracias al cual se pueden realizar todas las acciones que va a necesitar el cliente.

Para el almacenamiento de la información el servidor utiliza una base de datos MySQL a la que se conecta desde Java mediante JDBC. Toda la complejidad de la conexión y consulta a la base de datos se ha concentrado en una clase que proporciona dos métodos para realizar consultas (query y updateQuery) y que se encarga también de la reconexión en caso de perderse la conexión con la base de datos.

Además, nuestro servidor también actua como un servidor de almacenamiento de imágenes, proporcionando unos métodos mediante los cuales un cliente puede guardar imágenes en el servidor devolviendosele un identificador con el cual podrá obtener posteriormente la imagen.

El servidor notifica a eventos al cliente mediante RMI. Durante el login, el cliente proporcionó un objeto mediante RMI que implementa IListen con el cual poder notificar los diferentes eventos que van ocurriendo.

El cliente está desarrollado utilizando JavaFX, una tecnología que está surgiendo y que está destinada a ser el sucesor de Swing. El cliente sigue una estructura bastante jerárquica en el sentido de que hay ciertos elementos de la interfaz que contienen a otros (los llamados "Controller"). WorldController es el que recibe los eventos del servidor (mediante una clase intermedia dependiente del tipo de conexión que tenemos actualmente) y se encarga de reenviarlo a los elementos que contiene que a su vez hacen lo mismo.

Debido a la lentitud de JTwitter, de la carga de fxml de JavaFX y del bloqueo al que se ve sometido la interfaz durante ese tiempo, las diferentes partes de la interfaz se cargan mediante carga bajo demanda (lazy-loading) para reducir así los tiempos sin respuesta de la interfaz gráfica.

Nuestro cliente ofrece la gran mayoría de las características de Twitter: publicación de tweets, hashtags, busquedas, acortamiento de URLs, menciones, eventos, protección de cuentas, geolocalización, mensajes privados...

Definición de la interfaz del sistema

Toda la interfaz del sistema ha sido documentada con Javadoc. Debido a la imposiblididad de meterlo dentro de la memoria lo hemos agregado dentro de la misma carpeta que contiene esta misma memoria.

Diagrama de clases

Debido a la gran cantidad de clases que contiene nuestro sistema se han añadido las dos imágenes con el diagrama de clases del servidor y del cliente en la misma carpeta en la que se encuentra esta memoria.

Opinión personal

El proyecto nos a parecido interesante ya que hemos trabajado con diversas tecnologias de la plataforma java, integrándolas para obtener un producto final allá de una práctica introductoria. También nos ha gustado la idea de integrar nuestra aplicación con librerias externas y nos ha parecido que Jtwitter aun está lejos de ser una opcion viable para una aplicación profesional. Por otra parte javaFX tiene cosas mejorables como el hecho de que la aplicación muchas veces queda bloqueada al cargar la interfaz, pero en general ofrece una forma fácil de crear interfaces para aplicaciones complejas con relativa facilidad.