Introducción a la Programación – UNQ – 2do semestre de 2012

Segundo parcial – Mini Tweeter

Aclaraciones:

- *Esta evaluación es a libro abierto. Se pueden usar todo lo visto en la práctica y en la teórica, aclarando la referencia.*
- *No se olvide de poner nombre, nro. de legajo, nro. de hoja y cantidad total de hojas en cada hoja.*
- *Le recomendamos leer el enunciado en su totalidad y organizar sus ideas antes de comenzar la resolución.*
- *Recuerde que la intención es medir cuánto comprende usted del tema. Por ello, no dude en escribir todo lo que sabe, en explicar lo que se propone antes de escribir código, en probar su código con ejemplos, etc.*

El presente parcial tiene por finalidad simular la operatoria de *Tweeter*, una de las plataformas sociales más populares de la actualidad. Esta plataforma consiste de una comunidad de *usuarios* que envían mensajes llamados *tweets*. Los usuarios tienen *seguidores*, y son estos seguidores los que pueden ver los tweets de un usuario.

Para modelar la mencionada plataforma, se introducen las estructuras de datos que a continuación se describen (ver Fig. 1). Cada usuario se modela como un registro que contiene la identificación del mismo, la lista de tweets que envió hasta el momento y la lista de seguidores. Cada tweet viene dado por la fecha y hora en que se emitió y por el contenido en sí[1]. Finalmente, la plataforma en sí se modela como un registro que contiene a la lista de usuarios y a un número que servirá a la hora de dar de alta nuevos usuarios.

```
struct Usuario
{
    int miId;
    List<Tweet> misTweets;
    List<int> misSeguidores;
}

struct Tweet
{
    Fecha fecha;
    Hora hora;
    int msg;
}

struct Tweeter
{
    List<Usuario> usrs;
    int proximoId;
};
```

[1]Para simplificar, el contenido de un mensaje es un número en lugar de texto.

Se solicita resolver los siguientes ejercicios. Tener en cuenta que *ninguno* de ellos involucra el tablero de CGobstones.

Ejercicio 1 `Tweeter altaDeUsuario(Tweeter t)`

Propósito: *Da de alta un nuevo usuario. La identificación asignada al mismo está dada por el valor del campo* `proximoId`*. Este campo debe ser incrementado una vez que el alta se efectiviza, de modo tal que el próximo usuario que se agregue no repita ese número de identificación.*

Precondición: *No existe usuario en* `t` *con identificación* `proximoId`*.*

Ejercicio 2 `Tweeter bajaDeUsuario(Tweeter t, int id)`

Propósito: *Da de baja el usuario con identificación* `id`*. Además de eliminar el usuario de la lista de usuarios, tener en cuenta que* también *debe eliminarse* `id` *de la lista de los usuarios a los que sigue.*

Precondición: *El usuario* `id` *existe.*

Ejercicio 3 `Tweeter seguirA(Tweeter t, int id, int idASeguir)`

Propósito: *Retorna un nuevo tweeter en el que el usuario* `id` *ahora sigue al usuario* `idASeguir`*.*

Precondición: *Ambos usuarios existen y* `id` *no sigue a* `idASeguir`*.*

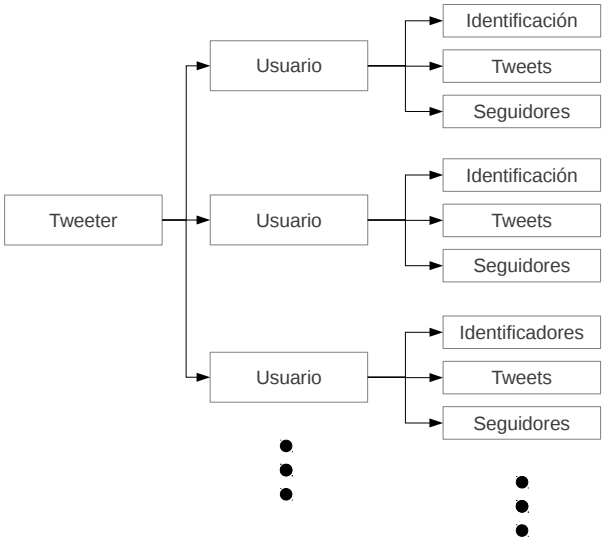

Figura 1: Diagrama de los principales elementos del modelo

Ejercicio 4 `Tweeter postTweet(Tweeter t, int idEmisor, Fecha f, Hora h, int msg)`

Propósito: *Agrega un nuevo tweet a la lista de tweets del usuario con identificación* `idEmisor`.

Precondición: *El usuario* `idEmisor` *existe.*

Ejercicio 5 `List<Tweet>tweetsVisiblesPorUsuario(Tweeter t, int id)`

Propósito: *Retorna la lista de todos los tweets que son visibles por el usuario* `id`. *Recordar que los tweets visibles son aquellos de todos los usuarios que* `id` *está siguiendo.*

Precondición: *El usuario* `id`*, como así también el de todos los usuarios que sigue, existe en* `t`.

Ejercicio 6 `int usuarioMasPopular(Tweeter t)`

Propósito: *Retorna la identificación del usuario más popular, es decir, aquel que tiene más seguidores.*

Precondición: *Hay al menos un usuario en* `t` *y además hay uno solo que es el más popular.*