Curso 2018-2019. Cuatrimestre Primavera. 12 de julio de 2019
Programación Orientada a Objetos. Examen Final de Reevaluación
Solución

Imaginad que se desea crear una versión propietaria de Twitter.

La fase inicial del análisis ha conducido al diagrama de clases incompleto (faltan métodos en las clases) que se muestra en la última página.

El programa debe gestionar **usuarios** que generarán **tweets**. A cada tweet pueden asociársele varios **hashtags**.

La aplicación distinguirá entre tweets originales y **retweets** de originales.

Un usuario podrá tener un grupo de seguidores (*followers*, en inglés). Al mismo tiempo, dicho usuario puede seguir (*following*) a un determinado grupo de otros usuarios. De cada usuario el programa guardará un identificador único, el nombre y una cierta información.

De cada tweet la aplicación guardará su texto y el instante de tiempo de su creación, expresado como un entero largo (tipo long de Java) igual a la cantidad de milisegundos pasados desde las 00:00 del 1 de Enero de 1970 UTC.

NOTA: el método estático de la clase System:

```
public static long currentTimeMillis();
```

devuelve justamente ese valor al ser invocado.

NOTA: Al resolver las preguntas del examen podéis suponer implementadas los métodos **getXXX() y setXXX()** que acceden a los atributos de las clases. **No es necesario que los implementéis**.

Ejercicio 1 **(1 punto).** A partir del diagrama UML mostrado, escribid para todas las clases, la parte de código de la definición que comienza con el "public class…." correspondiente a la declaración de los todos los atributos de las clases, incluyendo aquellos que aparecen al implementar las relaciones. NO implementéis todavía ningún método.

SOLUCIÓN

```
public class InterfazUsuario {
    private Controlador controlador;
…
}
public class Controlador {
    private InterfazUsuario iu;
    private Map<String, Usuario> usuarios;  //Clave: ID usuario
    private Map<String, HashTag> hashTags;  //Clave: Nombre del hashtag
…
}
public class Usuario {
    private String id, nombre, infoUser;
    private List<Tweet> muro;
    private Map<String, Usuario> following;
    private Map<String, Usuario> followers;
…
```

```
}
public class HashTag {
    private String nombre;
    private List<Tweet> tweets;
…
}
public class Tweet {

    protected long tPubMillis;
    protected String texto;
    protected Usuario autor;
    protected Map<String, HashTag> hashTags;    //Clave: Nombre del hashtag
…
}
public class Retweet extends Tweet {
    private Tweet original;
…
}
public class ReadUsuarioException extends Exception {
…
}
public class UsuarioNoExisteException extends Exception {
…
}
public class ReadUsuarioException extends Exception {
…
}
```

Ejercicio 2 **(1,5 puntos)**. Implementad en las clases Tweet y Retweet el método que se indica a continuación:

```
public String toString();
```

Este método debe generar un String de dos líneas. En la primera debe aparecer el nombre de usuario, el carácter '@', el identificador del usuario, el carácter '-' y una indicación del tiempo pasado desde que el tweet fue creado. Esa indicación contendrá el número de días pasados, si ha pasado 1 día o más. En caso de que no haya pasado ni un día y haya pasado una hora o más, contendrá el número de horas. Si no ha pasado ni una hora contendrá del número de minutos que hayan pasado (que puede ser 0). La segunda línea debe contener el texto del tweet.

Ejemplo: A continuación se muestra un tweet desde cuya creación han pasado 2,5 minutos:

```
PepeG@pepegId - 2 minutos
Texto del tweet
```

A continuación se muestra un tweet desde cuya creación han pasado 3,7 horas:

```
JorgeH@jorge_Id - 3 horas
Otro texto de tweet
```

SOLUCIÓN PARA CLASE Tweet

```
    public String toString()
    {
        long currentTime = System.currentTimeMillis();

        long ageMillis = currentTime - this.tPubMillis;
        long ageMins = ageMillis / 60000;
        long ageHours = ageMins / 60;
        long ageDays = ageHours / 24;

        StringBuilder s = new StringBuilder() ;
        s.append(this.autor.getNombre() + "@").append(this.autor.getId() + " -
");
        if (ageDays != 0)
            s.append(ageDays).append(" días\n");
        else if (ageHours != 0)
            s.append(ageHours).append(" horas\n");
        else
            s.append(ageMins).append(" minutos\n");

        s.append(this.texto);
        return s.toString();
    }
```

Para la clase Retweet el método además mostrará un texto que indique que es un retweet de otro tweet, e indicará el nombre, el carácter '@', el identificador del usuario que publicó el original. Por ejemplo:

```
JorgeH@jorge_Id - 0 minutos
Texto del tweet
Retweet de tweet originalmente publicado por PepeG@pepegId
```

SOLUCIÓN PARA CLASE Retweet

```
    public String toString()
    {
        StringBuilder res = new StringBuilder() ;
        String s = super.toString();
        res.append(s).append("\nRetweet de tweet publicado por ") ;
        res.append(this.original.autor.getNombre() + "@") ;
        res.append(this.original.autor.getId());
        return res.toString();
    }
```

Ejercicio 3 **(2 puntos)**. Implementad en la clase Usuario los siguientes métodos:

a) `public void publicaEnMuro(Tweet tweet);`

Este método debe añadir el tweet pasado a través del argumento `tweet` a la secuencia de tweets publicados por este usuario, como el primero de ellos.

```
public void publicaEnMuro(Tweet tweet)
{
    this.muro.add(0, tweet);
}
```

b) `public String getTextoMuro();`

Este método debe devolver un String que contenga la información de todos los tweets y retweets publicados en el muro del usuario. El texto de cada tweet/retweet se separará del siguiente por dos líneas en blanco. Si el usuario no ha generado ningún tweet/retweet, el String devuelto debe indicar que el muro del usuario está vacío.

```
public String getTextoMuro()
{
    StringBuilder s = new StringBuilder();

    if (this.muro.isEmpty())
        s.append("Muro vacío.");
    else
    {
        for (Tweet t : this.muro)
            s.append(t.toString() + "\n\n") ;
    }

    return s.toString();
}
```

c) `List<Usuario> getFollowingAndFollowers();`

Este método devuelve una lista de aquellos usuarios que son, a la vez, seguidores de este usuario y usuarios seguidos por este usuario.

```
public List<Usuario> getFollowingAndFollowers()
{
    //Devuelve una lista qcon aquellos usuarios que sigue
    //el mismo usuario y que también lo siguen a él

    List<Usuario> usuarios = new LinkedList<>();

    for (Usuario u : this.following.values())
    {
        if (this.getFollowers().containsKey(this.id))
            usuarios.add(u);
    }
    return usuarios;
}
```

Ejercicio 4 **(4 puntos)**. Implementad en la clase Controlador los siguientes métodos:

a) `public Set<String> getHashTags(String texto);`

Este método devuelve un conjunto cuyos componentes son todos los hashtags detectados en el argumento `texto`. El contenido de dicho argumento es un String en el que las palabras están separadas exclusivamente por espacios en blanco. El método debe considerar que un hashtag es una palabra que comienza con el carácter '#'. Un carácter '#' aislado no debe considerarse como un indicador de hashtag.

NOTA: podéis hacer uso del siguiente método de la clase String, cuyos detalles se copian de la documentación del Java:

```
public String substring(int beginIndex);
```

Returns a string that is a substring of this string. The substring begins with the character at the specified index and extends to the end of this string.

```
    public Set<String> getHashTags(String texto) {
        Set<String> hashTags = new HashSet<>();
        String[] partes = texto.split(" ");

        for (int i = 0; i < partes.length; i++) {
            if (partes[i].startsWith("#")) {
                //Es un hashtag
                String nombre = partes[i].substring(1);
                if (nombre.length() > 0) {
                    hashTags.add(nombre);
                }
            }
        }
        return hashTags;
    }
```

b) `public void nuevoTweet (String id, String texto) throws UsuarioNoExisteException, NoHashTagException`

Este método intenta que el usuario cuyo identificador coincide con el valor del argumento `id`, cree un nuevo tweet cuyos detalles (excepto la fecha de publicación contada en milisegundos a partir del 00:00 del 1 de Enero de 1970 UTC), se pasan en el argumento `texto`.

Si no existe ningún usuario cuyo identificador coincida con el valor de `id` el método debe lanzar la excepción `UsuarioNoExisteException`.

Si en el argumento `texto` no hay ningún hashtag, el método debe lanzar la excepción `NoHashTagException` (se asume que todo tweet debe asignársele como mínimo un hashtag).

En ninguno de los dos casos anteriores el método creará ningún tweet.

En caso de que no se cumplan las dos condiciones anteriores, el método creará un tweet cuyo texto será el valor pasado en el argumento `texto`. El método debe crear asimismo todas las asociaciones en las que el nuevo tweet debe verse involucrado según el diagrama de clases UML.

```
    public Tweet nuevoTweet(String id, String texto) throws
UsuarioNoExisteException,
            NoHashTagException {
```

```
        Usuario autor = this.usuarios.get(id);
        if (autor == null) {
            throw new UsuarioNoExisteException();
        }

        Set<String> htags = this.getHashTags(texto);
        if (htags.isEmpty()) {
            throw new NoHashTagException();
        }

        Tweet tweet = new Tweet(texto, autor);

        for (String nombre : htags) {
            HashTag ht = this.hashTags.get(nombre);
            if (ht == null) {
                ht = new HashTag(nombre);
                this.hashTags.put(nombre, ht);
            }
            //Añadimos hashtag a tweet
            tweet.getHashTags().put(nombre, ht);

            //Añadimos tweet a hashtag
            ht.getTweets().add(tweet);
        }
        autor.publicaEnMuro(tweet);
        return tweet;
    }
```

c) `public HashTag getTrendingTopicUltimaHora();`

Este método devuelve aquel objeto HashTag que haya sido mencionado en más tweets durante la última hora ("trending topic" en terminología inglesa).

```
    public HashTag getTrendingTopicUltimaHora() {
        long currentTime = System.currentTimeMillis();

        HashTag trending = null;
        int maxTweets = 0;

        for (HashTag ht : this.hashTags.values()) {
            int numTweets = 0;

            for (Tweet t : ht.getTweets()) {
```

```
                if (currentTime - t.getTPubMillis() < 60 * 60 * 1000) {
                    //Es un tweet publicado durante la ultima hora!
                    numTweets++;
                }
            }
            if (numTweets > maxTweets) {
                trending = ht;
                maxTweets = numTweets;
            }
        }
        return trending;
    }
```

d) `public List<Usuario> getUsuariosOrdenadosPorFollowersDecrecientes();`

Este método devuelve una lista con todos los usuarios de la aplicación, en donde estos queden ordenados por su número de seguidores (*followers*), de forma decreciente. Es decir, en la primera posición de la lista deberá aparecer el usuario más seguido y en la última aquél menos seguido.

```
    public List<Usuario> getUsuariosOrdenadosPorNumeroDeFollowersDecreciente() {
        List<Usuario> usuariosOrden = new ArrayList<>();

        for (Usuario u : this.usuarios.values()) {
            boolean insertado = false;
            int numFollowers = u.getFollowers().size();

            for (int i = 0; i < usuariosOrden.size() && !insertado; i++) {
                if (usuariosOrden.get(i).getFollowers().size() <= numFollowers)
{
                    usuariosOrden.add(i, u);
                    insertado = true;
                }
            }

            if (!insertado) {
                usuariosOrden.add(u);   //Añadimos al final (puede estar vacía
aún)
            }
        }

        return usuariosOrden;
    }
```

Ejercicio 5 **(1,5 puntos)**. Implementad en la clase Controlador el siguiente método:

```
public void readUsuarios(String fName) throws ReadUsuarioException;
```

Este método trata de leer usuarios presentes en un archivo de texto cuyo nombre (*path*) completo se pasa a través del argumento `fName`. Tened además en cuenta que:

- Se supone que el archivo NO contiene detalles de ningún retweet. Se supone asimismo que el archivo NO contiene ningún error de formato
- El método debe lanzar una excepción `ReadUsuarioException` si el archivo de texto no existe
- El método debe crear los usuarios y tweets correspondientes a partir del contenido del archivo
- El método debe añadir los usuarios creados al contenedor de usuarios del controlador
- El método debe añadir los hashtags creados al contenedor de hashtags del controlador
- El método debe crear todas las asociaciones especificadas en el diagrama de clases UML para tweets, usuarios y hashtags

A continuación, se muestra un ejemplo de la parte del fichero que contiene la información correspondiente a un usuario que haya publicado dos tweets:

```
Usuario
<id del usuario>;<nombre del usuario>
<información del usuario>
Tweet
<Número de milisegundos transcurridos desde las 00:00 del 1 de Enero de 1970 UTC hasta el instante en que el tweet se creó>
<Texto del tweet con todas las palabras separadas por espacios en blanco incluidos hashtags antecedidos por el carácter '#'>
Tweet
<Número de milisegundos transcurridos desde las del 1 de Enero de 1970 UTC hasta el instante en que el tweet se creó>
<Texto del tweet con todas las palabras separadas por espacios en blanco incluidos hashtags antecedidos por el carácter '#'>
FinUsuario
```

El archivo contiene tantas secciones que comienzan con la línea "Usuario" y acaban con la línea "FinUsuario", como usuarios deben crearse.

En cada una de esas secciones habrá tantas subsecciones que comiencen con la línea "Tweet" como tweets haya generado el usuario en cuestión.

NOTA: se sugiere hacer uso del método `nuevoTweet` solicitado en la pregunta 3 aunque no se haya implementado.

```
public void readUsuarios(String fName)
        throws ReadUsuarioException {
    try {
        FileInputStream fin = new FileInputStream(fName);
        Scanner scan = new Scanner(fin);
        String linea;
        while (scan.hasNextLine()) {
            linea = scan.nextLine();
            if (linea.length() > 0) {
                if (linea.equals("Usuario")) {
```

```
                    Usuario us = this.createUsuario(scan);
                    this.usuarios.put(us.getId(), us);
                    this.addTweets(us, scan);
                }
            }
        }
        scan.close() ;
    } catch (FileNotFoundException ex) {
        throw new ReadUsuarioException("Archivo no encontrado. Detalles: "
                + ex.getMessage());
    }
}
private Usuario createUsuario(Scanner scan) {
    String linea = scan.nextLine();
    String[] partes = linea.split(";");
    linea = linea = scan.nextLine();
    return new Usuario(partes[0], partes[1], linea);
}
private void addTweets(Usuario us, Scanner scan) throws ReadUsuarioException
{
    String linea = " ";
    long tmillis = 0;
    boolean fin = false;
    while (scan.hasNextLine() && !fin) {
        linea = scan.nextLine();
        if (linea.equals("FinUsuario")) {
            fin = true;
        }
        if (!fin) {
            tmillis = scan.nextLong();
            scan.nextLine();
            linea = scan.nextLine();
            try {
                Tweet tweet = this.nuevoTweet(us.getId(), linea);
                tweet.setTPubMillis(tmillis);
            } catch (Exception ex) {
                throw new ReadUsuarioException(ex.getMessage());
            }
        }
    }
}
```

DIAGRAMA DE CLASES UML DE LA APLICACIÓN: