Práctica 5: Colecciones

Objetivos

- Uso de colecciones.
- Documentación del código.

Introducción

En la práctica anterior se modeló un usuario de nuestra red social, y se gestionaron los mensajes que el usuario publicaba a largo del tiempo realizando operaciones de búsqueda sencillas. En esta práctica vamos a mejorar nuestro modelo del usuario para poder hacer un adecuado seguimiento de sus relaciones sociales, es decir, del conjunto de sus amigos (otros usuarios de nuestro sistema a los que este usuario sigue).

Actividades a desarrollar

Cree la clase Usuario en el paquete es.upm.dit.prog.p5 de su proyecto p5. Esta clase gestiona un usuario de nuestra red social, con las siguientes características:

- Un usuario dispone de un identificador de usuario único, que es una cadena de texto.
- Un usuario puede tener 0 o más mensajes en su *timeline,* que deberán estar ordenados según el momento en el que fueron publicados.
- Un usuario puede tener 0 o más amigos, que son otros *Usuario* de la red social. Un usuario no puede tener al mismo *Usuario* como amigo más de 1 vez, ni puede tener a un usuario *null* como amigo.

Para almacenar y gestionar las relaciones sociales y los mensajes de forma más eficiente, el alumno deberá utilizar las colecciones que estime más conveniente (Revisar la documentación de Java para saber qué tipo de colección se debe emplear, y los métodos disponibles que hacen de forma automática muchas de las funciones requeridas en esta práctica). No se podrán usar arrays durante el desarrollo de la práctica.

Además de estas características, la clase *Usuario* dispondrá de los siguientes métodos públicos que el alumno debe desarrollar (Puede implementar los métodos privados que considere oportunos):

Constructor. Inicia el estado del objeto. Toma como parámetro una cadena de texto que representa el identificador del usuario en la red social.

Excepciones: Debe lanzar una excepción del tipo ParametroErroneoException cuando el valor pasado como parámetro sea nulo o una cadena vacía.

setAmigos. Recibe como parámetro un conjunto de usuarios (Set <Usuario>) que reemplaza los amigos almacenados por los recibidos, eliminando todos los que estuvieran previamente almacenados. Si la colección pasada contiene algún valor null éste deberá ignorarse.

Excepciones: Debe lanzar una excepción del tipo ParametroErroneoException cuando el valor pasado como parámetro sea null o cuando el usuario intente ser amigo de sí mismo. En estos casos no se reemplazan los amigos que estuvieran almacenados previamente.

setAmigos. Recibe como parámetro una lista de usuarios (List<Usuario>) que reemplaza a los

amigos almacenados por los recibidos, eliminando todos los que estuvieran previamente almacenados. Si la colección pasada contiene algún valor null éste deberá ignorarse. Si la colección contiene el mismo amigo varias veces, sólo se deberá añadir ese amigo una vez.
Excepciones: Debe lanzar una excepción del tipo ParametroErroneoException cuando el valor pasado como parámetro sea null o cuando el usuario intente ser amigo de sí mismo, en cuyo caso no se reemplazan los amigos que estuvieran almacenados previamente.
getAmigos. Devuelve una colección (Set<Usuario>) que incluye todas los amigos de este usuario. La colección devuelta no puede contener elementos nulos. Si no hay datos almacenados deberá devolver un conjunto vacío.
addMensaje. Recibe como parámetro un mensaje (Mensaje) para ser almacenado en el timeline del usuario. Este método no devuelve nada.
Excepciones: Debe lanzar una excepción del tipo ParametroErroneoException cuando el valor pasado como parámetro sea null.
setMensajes. Recibe como parámetro una lista de mensajes (List<Mensaje>), que reemplaza a los mensajes almacenados por los recibidos, eliminando todos los que estuvieran previamente almacenados. Si la colección pasada contiene algún valor null éste deberá ignorarse. No se conoce si los mensajes recibidos están ordenados temporalmente o no.
Excepciones: Debe lanzar una excepción del tipo ParametroErroneoException cuando el valor pasado como parámetro sea null.
getMensajes. No toma parámetros. Devuelve una colección List que incluye todas los mensajes de este usuario (List<Mensaje>). La lista devuelta debe estar ordenada temporalmente siendo el mensaje más reciente el almacenado en la primera posición de la lista. La colección devuelta no puede contener elementos nulos. Si no hay datos almacenados deberá devolver una lista de 0 elementos.
getMensajes. Recibe dos parámetros, de tipo Calendar, que acotan el periodo de búsqueda de mensajes de este usuario. El primero indica la fecha desde la que se empieza a buscar y el segundo indica la fecha en la que se termina de buscar, de forma que la primera fecha debería ser anterior a la segunda fecha. Devuelve una colección List que incluye todos los mensajes (List<Mensaje>) que este usuario ha publicado entre esas dos fechas. La lista devuelta debe estar ordenada temporalmente siendo el mensaje más reciente el almacenado en la primera posición de la lista. La colección devuelta no puede contener elementos nulos. Si no hay datos almacenados deberá devolver una lista de 0 posiciones.
Excepciones: Debe lanzar una excepción del tipo ParametroErroneoException cuando algún valor pasado como parámetro sea null, o si la primera fecha no es anterior a la segunda.
getMensajesDeAmigos. Recibe un parámetro, de tipo Calendar, que indica una fecha. Devuelve una colección List que incluye todos los mensajes (List<Mensaje>) que los amigos del usuario han publicado con posterioridad a esa fecha (desde esa fecha hasta este momento). La lista devuelta debe estar ordenada temporalmente siendo el mensaje más reciente el almacenado en la primera posición de la lista. La colección devuelta no puede contener elementos nulos. Si no hay datos almacenados deberá devolver una lista de 0 posiciones.
Excepciones: Debe lanzar una excepción del tipo ParametroErroneoException cuando el valor pasado como parámetro sea null.
equals. Compara un usuario pasado como parámetro con este usuario. Devuelve true si los

nombres de los usuarios coinciden, y false en caso de que el parámetro pasado sea null o los nombres de los usuarios sean distintos.

El alumno debe documentar todo el código de la clase *Usuario*.

Pruebas de la clase Usuario

En esta ocasión se proporciona al alumno un fichero ***Corrector.java*** que incluye un conjunto exhaustivo de pruebas JUnit que se realizan a la clase Usuario. El alumno deberá incluir este fichero en su proyecto, y probar que su implementación de la clase Usuario supera todas las pruebas. La evaluación de la entrega del alumno se hará usando estas mismas pruebas.

Documentos y ficheros proporcionados

El código de las clases Mensaje y ParametroErroneoException podrá encontrarlo en el Moodle de la asignatura.

Evaluación

La evaluación de la práctica se basará en los siguientes aspectos:

- **Corrección de Usuario.java**: A la clase desarrollada por el alumno se le pasará una batería de pruebas para comprobar su corrección funcional. La clase desarrollada debe cumplir la especificación de la clase referente a las funciones a realizar y a las excepciones que se deben lanzar (**8 puntos**).
- **Estilo y documentación**: Se revisará el seguimiento de las directrices de estilo y la documentación empleando comentarios javadoc. Se deberá documentar la clase, los métodos y los atributos (**2 puntos**). Esta parte no será evaluada si la clase *Usuario* no alcanza un mínimo de 6 puntos sobre 8 en la evaluación.

Entrega de la práctica

La práctica se deberá entregar en Moodle antes de las 23:55 del día 30 de mayo de 2014. En esta ocasión el corrector está configurado para **limitar a 3 el número de veces que un alumno puede comprobar su calificación**. Una vez que el alumno haya realizado 3 envíos el alumno no podrá realizar nuevas entregas. La calificación final de cada alumno será aquella que obtuvo en el último envío que realizó.

El fichero entregado debe cumplir con los siguientes requisitos:

1. Su nombre debe ser “practica5.zip”.
2. Debe incluir el código fuente (fichero .java) de la clase *Usuario*.
3. Los ficheros .java deben estar en el directorio relativo: “**p5/src/es/upm/dit/prog/p5/**”.

NOTA IMPORTANTE: Si no se cumplen los requisitos de nombres de ficheros y estructura de directorios, o si la práctica no compila, la práctica no se evaluará.

Se recuerda a los alumnos que la copia de código supone no sólo el suspenso automático de la práctica, sino también en la asignatura, tanto

para quien copia el código como para quien lo cede, sin distinción. Además, se emprenderán medidas disciplinarias contra dichos alumnos ante la Escuela y la Universidad.