

EXAMEN PARCIAL DE PROGRAMACIÓN ORIENTADA A OBJETOS (POO)
CURSO 2017/2018. CUATRIMESTRE DE PRIMAVERA. 2 DE MAYO DE 2018.

El diagrama de clases UML de la figura mostrada en la **última hoja del examen** muestra el núcleo de una aplicación del tipo Instagram.

Los usuarios suscritos a esta aplicación disponen de un muro en el que pueden publicar fotografías o vídeos cortos de unos segundos de duración. A cada publicación (modelada por la clase Post en el diagrama UML) el usuario puede asociarle una descripción y una serie de descriptores (conocidos en terminología inglesa como “hashtags”; un descriptor se modela con la clase HashTag en el diagrama UML), que sirven para facilitar la búsqueda de publicaciones relacionados con una cierta temática identificada por dichos descriptores.

La aplicación debe diseñarse de forma que guarde memoria de todos los descriptores asociados por los usuarios a sus publicaciones. La operación más frecuente realizada sobre estos descriptores será simplemente la de comprobar si un descriptor elegido por un cierto usuario ya había sido asociado previamente a otra publicación.

Todo usuario podrá seguir a uno o varios usuarios, y podrá ser seguido por uno o varios usuarios (observad la asociación bidireccional de muchos a muchos que conecta la Usuario consigo misma). Para convertirse en seguidor de otro, un usuario debe generar una petición de seguimiento que el usuario seguido deberá aceptar.

Las publicaciones realizadas en el muro de un usuario que configura dicho muro como público serán accesibles a todos los usuarios de la aplicación. Las realizadas en el muro de un usuario que lo configura como privado, solo serán accesibles a los seguidores de ese usuario.

Todo usuario puede valorar cualquier publicación a la que pueda acceder. Al valorarla, el usuario puede declarar que la publicación “le gusta”, o añadir un comentario, o ambas cosas simultáneamente. Cada valoración realizada por un usuario generará un objeto Valoracion convenientemente asociado a la publicación a la que valora y al usuario que la ha generado.

NOTA 1: Al resolver las preguntas del examen podéis suponer implementadas los métodos inmediatos getXXX() y setXXX() que acceden a los atributos de las clases. **No es necesario que los implementéis a no ser que se indique lo contrario.**

NOTA 2: Si durante la resolución de las preguntas necesitáis de métodos más complejos que los de acceso directo a los atributos de una clase mencionados en la nota anterior, **DEBERÉIS IMPLEMENTAR DICHS MÉTODOS.**

PREGUNTA 1 (1,5 puntos). Para todas las clases del diagrama UML, escribid la parte de código de la definición que comienza con el “public class...” correspondiente solo a la declaración de los todos los atributos de las clases, incluyendo aquellos que aparecen al implementar las relaciones mostradas en el diagrama.

```
public class InstantP00
{
    private Map<String, Usuario> usuarios; //Clave: userID
    private List<PeticiónSeguimiento> pendientes;
    private Map<Integer, Post> posts; //Clave: identificador
    private Map<String, HashTag> hashtags; //Clave: nombre
```

```

}

public class Usuario
{
    private String userID, nombre, email;
    private boolean muroPrivado;
    private Map<String, Usuario> seguidores; //Clave: userID
    private Map<String, Usuario> siguiendo; //Clave: userID
    private List<PeticionSeguimiento> pendientes;
    private List<Post> muro;
}

public class PeticionSeguimiento
{
    private Usuario creador;
    private Usuario aSeguir;
}

public abstract class Post
{
    protected int identificador;
    protected String fecha, descripcion, nombreArchivo;
    protected Usuario autor;
    protected Map<String, HashTag> hashtags; //Clave: nombre
    protected List<Valoracion> valoraciones;
}

public class Foto extends Post
{
}

public class Video extends Post
{
    public static final int MAX_DURACION = 10;
    private int duracion;
}

public class HashTag
{
    private List<Post> posts;
}

public class Valoracion
{
    private boolean like;
    private String comentario;
    private Usuario autor;
}

```

PREGUNTA 2 (1 punto). Implementad los siguientes constructores:

1. El de la clase **Post**. Decidid, a la vista de la semántica de los diferentes atributos, los argumentos que deben ser pasados al constructor. Justificadlo muy brevemente.

Nota: considerad que cuando se crea una nueva publicación (Post), ya se conocen los descriptores (hashtags) que deben asociársele.

2. De **Video**.

```

public abstract class Post
{

```

```

    public Post(int identificador, String fecha, String descripcion, String
nombreArchivo, Usuario autor, Map<String, HashTag> hashtags)
    {
        this.identificador = identificador;
        this.fecha = fecha;
        this.descripcion = descripcion;
        this.nombreArchivo = nombreArchivo;
        this.autor = autor;
        this.hashtags = hashtags;
        this.valoraciones = new ArrayList<>();
    }
}

public class Video extends Post
{
    public Video(int identificador, String fecha, String descripcion, String
nombreArchivo, Usuario autor, Map<String, HashTag> hashtags, int duracion)
    {
        super(identificador, fecha, descripcion, nombreArchivo, autor, hashtags);
        this.duracion = duracion;
    }
}

```

PREGUNTA 3 (1 punto). Implementad en la clase **Valoracion** el método:

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

Este método debe devolver un String con los detalles de un comentario, incluyendo los el identificador del usuario que lo ha generado.

Implementad en la clase **Post** el método:

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

Este método debe devolver un String con los detalles de una cierta publicación, incluyendo todos sus descriptores y comentarios (para cada comentario también debe incluir el identificador del usuario que lo ha generado).

```

public class Valoracion
{
    public String toString()
    {
        String s = "Autor: " + this.autor.getUserID() + "\n";
        if (this.like)
            s = s + "Like!";

        s = s + "\n" + this.comentario;
        return s;
    }
}

public abstract class Post
{
    public String toString()
    {
        String s = "Identificador: " + this.identificador + "\n";
        s = s + "Fecha: " + this.fecha + "\n";
        s = s + "Descripcion: " + this.descripcion + "\n";
        s = s + "Nombre archivo: " + this.nombreArchivo + "\n";
        s = s + "Autor: " + this.autor.getUserID() + "\n";
        s = s + "HashTags: \n";
    }
}

```

```

        for (String nombre: this.hashtags.keySet())
        {
            s = s + "#" + nombre + "\n";
        }

        for (Valoracion v : this.valoraciones)
        {
            s = s + v.toString() + "\n";
        }

        return s;
    }
}

```

PREGUNTA 4 (2 puntos). Implementad los métodos de la clase **InstantPOO**:

```

public PeticionSeguimiento creaPeticionSeguir(String idUsSeguidor, String
idUsSeguido);

```

Este método trata de crear un nuevo objeto PeticionSeguimiento. El argumento idUsSeguidor se corresponde con el identificador del usuario que genera la petición para seguir a otro. El argumento idUsSeguido se corresponde con el identificador del usuario al que el que ha generado la petición pretende seguir. Si alguno de los argumentos (o ambos) no se corresponde con ningún identificador de usuario de la aplicación, el método no podrá crear el objeto PeticionSeguimiento.

Si el método puede crear un nuevo objeto PeticionSeguimiento, debe asociar dicho objeto al usuario que quiere convertirse en seguidor y al usuario al que el seguidor quiere seguir.

Finalmente ese mismo objeto PeticionSeguimiento debe añadirse al contenedor de peticiones de seguimiento pendientes de procesar.

El método devolverá el nuevo objeto PeticionSeguimiento si ha podido generarlo sin problemas. Devolverá null en caso contrario.

```

public boolean aceptaPeticionSeguir(String userID, String userIDSeguidor;

```

Este método intentará procesar, de entre todas las peticiones de seguimiento pendientes de procesar que tiene la aplicación, aquella que tiene como seguidor el usuario cuyo identificador se pasa en el argumento userIDSeguidor y como usuario seguido el usuario cuyo identificador se pasa en el argumento userID.

En caso de que alguno o ambos identificadores no se corresponda con un usuario, el método debe devolver false.

En caso de poder procesar la petición con éxito, el método debe devolver true. Pero antes de devolver dicho valor el método debe hacer que los dos usuarios involucrados en la petición queden asociados el uno con el otro (observad la asociación bidireccional que entra y sale de Usuario en el diagrama UML), esto es, que en el usuario seguidor se realice la anotación del usuario al que se comienza a seguir, y que en el usuario seguido se realice la anotación del nuevo seguidor. Cuando estas asociaciones se hayan realizado, la petición debe ser eliminada del contenedor de peticiones pendientes de procesar.

```

public class InstantPOO
{
    public PeticionSeguimiento creaPeticionSeguir (String userIDSeguidor,
String userIDSeguir)
    {
        Usuario seguidor = this.usuarios.get(userIDSeguidor);

```

```

        Usuario aSeguir = this.usuarios.get (userIDSeguir);

        if (seguidor == null || aSeguir == null)
            return null;

        PeticionSeguimiento p = new PeticionSeguimiento(seguidor, aSeguir);
        aSeguir.getPendientes().add(p); //Pendientes de usuario
        this.pendientes.add(p); //Pendientes (todas) de InstantP00
        return p;
    }

    public boolean aceptaPeticionSeguir(String userID, String userIDSeguidor)
    {
        Usuario usuario = this.usuarios.get(userID);
        Usuario seguidor = this.usuarios.get (userIDSeguidor);

        if (usuario == null || seguidor == null)
            return false;

        PeticionSeguimiento procesada = null;
        for (PeticionSeguimiento p : this.pendientes)
        {
            if (p.getASeguir() == usuario && p.getSeguidor() == seguidor)
            {
                procesada = p;
                break;
            }
        }

        if (procesada != null)
        {
            usuario.getSeguidores().put(userIDSeguidor, seguidor);
            seguidor.getSiguiendo().put(userID, usuario);
            usuario.getPendientes().remove(procesada); //Pendientes de usuario
            this.pendientes.remove(procesada); //Pendientes (todas) de InstantP00
            return true;
        }
        else
        {
            return false;
        }
    }
}

```

PREGUNTA 5 (1,5 puntos). Implementad el método de la clase **Post**:

```
public List<Usuario> getUsuariosLesGusta();
```

Este método devuelve una nueva lista de usuarios. Esta lista contiene todos los objetos Usuario asociados a objetos Valoracion que tienen una indicación de “me gusta” en su interior (atributo like con valor “true”). En otras palabras, el método devuelve una lista con todos los usuarios a quienes esta publicación les gusta.

NOTA: la lista no debe contener usuarios repetidos (un mismo usuario puede haber escrito más de una valoración de una publicación). Recordad que las listas tienen un método contains(E e).

```

public abstract class Post
{
    public List<Usuario> getUsuariosLesGusta()
    {
        List<Usuario> usuariosGusta = new ArrayList<>();
    }
}

```

```

        for (Valoracion v : this.valoraciones)
        {
            if(v.getLike())
            {
                Usuario autor = v.getAutor();
                if (!usuariosGusta.contains(autor))
                    usuariosGusta.add(autor);
            }
        }

        return usuariosGusta;
    }
}

```

PREGUNTA 6 (1,5 puntos). Implementad el método de la clase **Post**:

```
public boolean haSidoValoradoPorUsuario (Usuario us);
```

Este método devuelve true si la publicación (Post) donde se define el método ha sido valorado por el usuario pasado como argumento. Devuelve false en caso contrario

Implementad el método de la clase **InstantPOO**:

```
public List<Post> getPostsValorados (Usuario us);
```

Este método devuelve una nueva lista de publicaciones. Dicha lista debe contener todas las publicaciones valoradas por este usuario (el usuario que es gestionado por el objeto que contiene al método).

```

public abstract class Post
{
    public boolean haSidoValoradoPorUsuario (Usuario us)
    {
        for (Valoracion v : this.valoraciones)
        {
            if (v.getAutor() == us)
                return true;
        }

        return false;
    }
}

public class InstantPOO
{
    public List<Post> getPostsValorados (Usuario us)
    {
        List<Post> valorados = new ArrayList<>();
        for (Post p : this.posts.values())
        {
            if (p.haSidoValoradoPorUsuario(us))
                valorados.add(p);
        }

        return valorados;
    }
}

```

PREGUNTA 7 (1,5puntos). Implementad el método de la clase **InstantPOO**:

```
public List<Post> getPublicacionesConAlgunHashTag ( List<HashTag> hashtags);
```

Este método devuelve una nueva lista de publicaciones. Dicha lista debe contener todas las publicaciones que tienen asociado **al menos un descriptor** ("hashtag") idéntico a uno de los descriptores (hashtags) pasados en la lista de descriptores a través del argumento hashtags.

```
public List<Post> getPublicacionesConTodosLosHashTags (List<HashTag>
hashtags);
```

Este método devuelve una nueva lista de publicaciones. Dicha lista debe contener todas las publicaciones que tienen asociados **todos los descriptores** ("hashtag") pasados en la lista de descriptores a través del argumento hashtags.

```
public class InstaP00
{
    public List<Post> getPublicacionesConAlgunHashTag (List<HashTag> hashtags)
    {
        List<Post> lista = new ArrayList<>();
        for (Post p : this.posts.values())
        {
            for (HashTag h : hashtags)
            {
                if(p.getHashtags().containsKey(h.getNombre()))
                {
                    lista.add(p);
                    break;
                }
            }
        }

        return lista;
    }

    public List<Post> getPublicacionesConTodosLosHashTags (List<HashTag>
hashtags)
    {
        List<Post> lista = new ArrayList<>();
        for (Post p : this.posts.values())
        {
            boolean todos = true;
            for (HashTag h : hashtags)
            {
                if(!p.getHashtags().containsKey(h.getNombre()))
                {
                    todos = false;
                }
            }

            if(todos)
                lista.add(p);
        }

        return lista;
    }
}
```


Diagrama de clases UML. NOTA: el símbolo # indica que el atributo o método al que afecta es “protected”.

