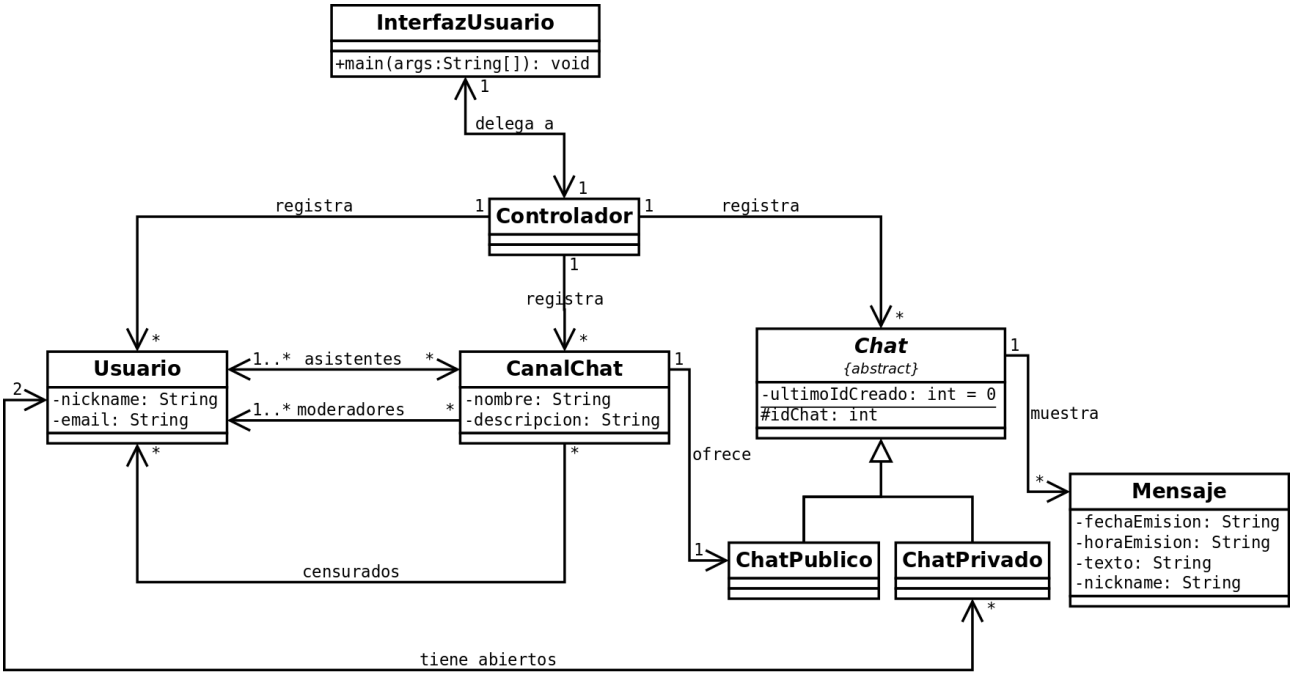
Curso 2016-2017. Cuatrimestre de primavera. 8 de Junio de 2017
Programación Orientada a Objetos (POO). Examen final. Duración del examen: 2h 30min
Publicación de notas provisionales: 19/6/2017 (mediodía).
Revisión de examen:20/6/2017. Hora: 11:00 Lugar: C6-E106

El diagrama de clases UML (incompleto en términos de métodos de las clases) que se muestra a continuación pretende ser el correspondiente al núcleo de una aplicación de gestión de chats que permiten intercambiar mensajes instantáneos entre usuarios de la misma.



NOTA: PARA NO COMPLICAR EL DIAGRAMA, ÉSTE NO MUESTRA CLASES EXCEPCIÓN QUE SIN EMBARGO DEBEN CONSIDERARSE EXISTENTES AL RESPONDER A DETERMINADAS PREGUNTAS DEL EXAMEN.

Todo usuario dado de alta dispone de un apodo (llamado *nickname*) que es único. Una vez dado de alta, puede enviar mensajes a chats. Existen dos tipos de chats: públicos y privados. Todo chat tiene un identificador único que es un valor entero. **Asimismo, dispone de un atributo estático que indica cual es el identificador del último chat creado (ultimoIdCreado)**. Todo chat contiene una secuencia mensajes. Un mensaje incluye el texto enviado al chat, el apodo (nickname) del usuario que lo ha enviado y la fecha y hora a la que ha sido enviado.

Un usuario envía mensajes a un chat público a través de su canal de chat. Un canal de chat tiene un nombre único y una descripción y da acceso a un chat público. Un canal de chat es gestionado por uno o varios moderadores (que son también usuarios).

Los moderadores pueden censurar a determinados usuarios si lo consideran oportuno. Un usuario censurado por un moderador de un canal de chat deja de ser un asistente y pasa a formar parte del grupo de usuarios censurados en ese canal.

Dos asistentes a un chat público pueden abrir un chat privado (que solo tiene esos dos asistentes). Los mensajes enviados a ese chat no son vistos por ningún otro usuario de la aplicación.

Al resolver las preguntas del examen deberéis indicar qué operaciones getXX() y setXX(…) consideráis implementadas en las clases. No es necesario que las implementéis a no ser que se indique lo contrario.

PREGUNTA 1 (1,25 PUNTOS) A partir del diagrama de clases UML de la figura, escribid para las clases `Controlador`, `Usuario`, `CanalChat`, `Chat`, `ChatPublico`, `ChatPrivado` y `Mensaje` 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 mostradas en el diagrama. **NO debéis añadir nada más en el código de estas clases** en vuestras respuestas a esta pregunta.

```
public class Controlador {
    private InterfazUsuario iu;
    private Map<String, Usuario> usuarios;    //Clave: nickname
    private Map<String, CanalChat> canales;   //Clave: nombre del canal
    private Map<Integer, Chat> chats;         //Clave: idChat
}

public class Usuario {
    private String nickname, email;
    private Map<String, CanalChat> canales;          //Clave: nombre del canal
    private Map<Integer, ChatPrivado> chatsPrivados; //Clave: idChat
}

public class CanalChat {
    private String nombre, descripcion;
    private Map<String, Usuario> asistentes;    //Clave: nickname
    private Map<String, Usuario> moderadores;   //Clave: nickname
    private Map<String, Usuario> censurados;    //Clave: nickname
    private ChatPublico chatPublico;
}

public abstract class Chat {
    private static int ultimoIdCreado = 0;
    protected int idChat;
    protected List<Mensaje> mensajes;
}

public class ChatPublico extends Chat {
}

public class ChatPrivado extends Chat {
    private Usuario[] asistentes;
}

public class Mensaje {
    private String fechaEmision, horaEmision, texto, nickname;
}
```

PREGUNTA 2 (1 PUNTO) Proponed y diseñad un constructor para las clases `Chat`, `ChatPublico` y `CanalChat`. El constructor de la clase `Chat` debe calcular e inicializar el valor del identificador único del mismo.

```
public abstract class Chat {
    public Chat() {
```

```
        this.idChat = Chat.ultimoIdCreado+1;
        Chat.ultimoIdCreado = this.idChat;
        this.mensajes = new ArrayList<Mensaje>();
    }
}

public class ChatPublico extends Chat {
    public ChatPublico() {
        super();
    }
}

public class CanalChat {
    public CanalChat (String nombre, String descripcion) {
        this.nombre = nombre;
        this.descripcion = descripcion;
        this.asistentes = new HashMap<String, Usuario>();
        this.moderadores = new HashMap<String, Usuario>();
        this.censurados = new HashMap<String, Usuario>();
        this.chatPublico = new ChatPublico();
}
```

PREGUNTA 3 (1,5 PUNTOS) Implementad los métodos toString() de la clase `CanalChat` y `ChatPublico`. En ambos casos los métodos deben mostrar todos los detalles que sean accesibles desde los objetos de esas clases, incluyendo los mensajes, siguiendo el formato que se indica a continuación:

Chat: <idChat>

Asistentes: <nickName1>,<nickName2>, @<nickName3>, …

Mensajes:

[<fechaEmision> <horaEmision>] <nickName>: <texto>

Los apodos de los moderadores deben ir precedidos del carácter '@'. Lo encerrado entre < > indica que debe aparecer el valor del atributo indicado.

```
public class CanalChat {
    public String toString() {
        String s = "Chat: " + this.chatPublico.getIdChat() + "\n";

        s += "Asistentes: ";
        Iterator<Usuario> it = this.asistentes.values().iterator();
        while (it.hasNext()) {
            Usuario u = it.next();
            if (this.moderadores.contains(u))
                s += "@" + u.getNickname();
            else {
                s += u.getNickname();
            }

            if(it.hasNext())
                s += ", ";
        }
```

```
        s += "\nMensajes: \n";

        List<Mensaje> mensajes = this.chatPublico.getMensajes();
        for (Mensaje m : mensajes) {
            s += m.toString() + "\n";
        }

        return s;
    }
}

public class ChatPublico extends Chat {
    public String toString(){
        String s = "IdChat: " + this.idChat + "\n";
        s += "Mensajes: \n";
        for (Mensaje m : this.mensajes) {
            s += m.toString() + "\n";
        }
        return s;
    }
}

public class Mensaje {
    public String toString() {
        String s = "[" + this.fechaEmision + " " + this.horaEmision + "] ";
        s += this.nickname + " : " + this.texto;
        return s;
    }
}
```

PREGUNTA 4 (1,5 PUNTOS) Implementad en Controlador el método siguiente:

`public void accederAChat(String nombreCanal, String nickName)throws NoNickNameRegistradoException, UsuarioCensuradoException`

Este método se invoca cuando un usuario desea acceder a un cierto canal de chat. El argumento `nombreCanal` indica el nombre del canal de chat, y `nickName` indica el apodo del usuario.

Si no existe ningún usuario registrado con el apodo indicado lanza la excepción `NoNickNameRegistradoException`. En cambio, si el usuario con el apodo indicado está censurado en ese canal de chat lanza la excepción `UsuarioCensuradoException`.

El método añade el usuario a los asistentes del chat si todo va bien. Si no existe ningún canal con nombre igual al argumento `nombreCanal` el método crea un nuevo canal de chat (con descripción String vacío), crea su chat público correspondiente y añade al usuario con apodo igual al valor del argumento `nickName` como moderador del canal de chat.

```
public class Controlador {
    public void accederAChat (String nombreCanal, String nickName) throws
NoNickNameRegistradoException, UsuarioCensuradoException {

        Usuario u = this.usuarios.get(nickName);
        if (u == null)
            throw new NoNickNameRegistradoException();
```

```
        CanalChat canal = this.canales.get(nombreCanal);
        if (canal != null) {
            if (canal.getCensurados().containsKey(nickName)) {
                throw new UsuarioCensuradoException();
            }

            canal.getAsistentes().put(nickName, u);
            u.getCanales().put(canal.getNombre(), canal);   //Relación bidir.
        }
        else {
            canal = new CanalChat(nombreCanal, "");
            canal.getModeradores().put(nickName, u);
            u.getCanales().put(canal.getNombre(), canal);   //Relación bidir.
            this.canales.put(nombreCanal, canal);
        }
    }
}
```

PREGUNTA 5 (1 PUNTO) Implementad en Controlador el método siguiente:

`public void censurarUsuario(String nombreCanal,String nickNameACensurar, String nickNameModerador)throws NoNickNameRegistradoException, NoCanalException, NoAsistenteException, NoModeradorException`

Este método censura permite al moderador con apodo indicado en el argumento `nickNameModerador` censurar al usuario con apodo indicado en el argumento `nickNameACensurar` del canal de chat indicado en el argumento `nombreCanal`.

Si no existe ningún canal con ese nombre lanza la excepción `NoCanalException`. Si no existe ningún usuario registrado con los apodos indicados por los argumentos `nickNameACensurar` o `nickNameModerador` lanza la excepción `NoNickNameRegistradoException`. Si el argumento `nickNameACensurar` no identifica un asistente del canal, lanza la excepción `NoAsistenteException`. Si el argumento `nickNameModerador` no es un apodo de un moderador de ese canal de chat, lanza la excepción `NoModeradorException`.

```
public class Controlador {
    public void censurarUsuario(String nombreCanal,String nickNameACensurar,
String nickNameModerador)throws NoNickNameRegistradoException, NoCanalException,
NoAsistenteException, NoModeradorException {

        CanalChat canal = this.canales.get(nombreCanal);
        if (canal == null)
            throw new NoCanalException();

        Usuario aCensurar = this.usuarios.get(nickNameACensurar);
        Usuario moderador = this.usuarios.get(nickNameModerador);
        if (aCensurar == null || moderador == null)
            throw new NoNickNameRegistradoException();

        if (canal.getAsistentes().containsKey(nickNameACensurar) == false)
            throw new NoAsistenteException();

        if (canal.getModeradores().containsKey(nickNameModerador)== false)
```

```
            throw new NoModeradorException();

        canal.getCensurados().put(nickNameACensurar, aCensurar);
        canal.getAsistentes().remove(nickNameACensurar);
    }
}
```

PREGUNTA 6 (1,5 PUNTOS) Implementad en `Chat` el método siguiente:

```
public List<Mensaje> mensajesDeAsistente(String nickName)
```

Este método devuelve una lista con los mensajes escritos por el asistente con el apodo indicado por el argumento `nickName`.

```
public class Chat {
    public List<Mensaje> mensajesDeAsistente(String nickName){
        List<Mensaje> listaMensajes = new ArrayList<Mensaje>();
        for (Mensaje m : this.mensajes) {
            if (m.getNickname().equals(nickName))
                listaMensajes.add(m);
        }
        return listaMensajes;
    }
}
```

Implementad en Controlador el método siguiente:

```
public Chat chatConMasParticipación(String nickName) throws
NoNickNameRegistradoException
```

Este método devuelve el chat al que el usuario con el apodo indicado por el argumento `nickName` ha enviado un mayor número de mensajes. Debe devolver un solo chat independientemente de si son varios los chats a los que ha enviado este máximo número de mensajes. Si no existe ningún usuario registrado con el apodo indicado lanza la excepción `NoNickNameRegistradoException`.

```
public class Controlador {
    public Chat chatConMasParticipación(String nickName) throws
NoNickNameRegistradoException {
        if (this.usuarios.containsKey(nickName) == false)
            throw new NoNickNameRegistradoException();

        Chat chatMaxPart = null;
        int maxMensajes = 0;

        for (Chat chat : this.chats.values()) {
            int numMensajes = chat.mensajesDeAsistente(nickName).size();
            if (numMensajes > maxMensajes) {
               chatMaxPart = chat;
               maxMensajes = numMensajes;
            }
        }

        return chatMaxPart; //Devuelve null si no participa en ningún chat
    }
```

```
}
```

PREGUNTA 7 (1 PUNTO) Implementad la clase `ChatComparator`. Esta clase debe implementar el interface `java.util.Comparator<CanalChat>`.

Debéis por tanto definir y codificar un constructor adecuado y el método siguiente:

```
public int compare(CanalChat c1, CanalChat c2)
```

Este método devolverá un número positivo si el canal correspondiente al argumento c1 tiene más asistentes que el canal correspondiente al argumento c2, 0 si ambos tienen el mismo número de asistentes y un número negativo si el canal correspondiente al argumento c1 tiene menos asistentes que el canal correspondiente al argumento c2.

SUGERENCIA: Ayudaos de lo que hicisteis en el ejercicio evaluable 2 del laboratorio de la asignatura.

```
public class ChatComparator implements Comparator<CanalChat> {

    public ChatComparator() {}

    public int compare (CanalChat c1, CanalChat c2) {
        return c1.getAsistentes().size() – c2.getAsistentes().size();
    }
}
```

PREGUNTA 8 (1,25 PUNTOS) Implementad en Controlador el método siguiente:

```
public void escribirChatEnFichero(String nombreCanal, String fName, String
fechaInicial, String fechaFinal) throws IOException, NoCanalException
```

Este método intentará escribir los contenidos que fueron enviados al canal de chat identificado por el argumento `nombreCanal` entre las fechas identificadas por los argumento `fechaInicial` y `fechaFinal` ambas incluidas (podéis suponer que se codifica una fecha mediante un String cuyo contenido sigue el patrón "DD/MM/AAAA"), en el fichero cuyo nombre completo es igual al argumento `fName`.

Si no hay canal con nombre igual a `nombreCanal` lanzará una excepción `NoCanalException`. Si hay algún problema con alguna operación relacionada con el fichero lanzará una excepción `IOException`.

Para resolver este problema podéis suponer que existe una clase `FechaUtils`, que no aparece en el diagrama de clases, que contiene el método siguiente:

```
public int compareFechas(String unaFecha, String otraFecha)
```

Este método devuelve un número positivo si la fecha representada por el argumento `unaFecha` (según el patrón indicado anteriormente para el argumento `fecha` del método que tenéis que construir) es posterior a la fecha representada por `otraFecha`, devuelve 0 si ambos argumentos representan la misma fecha, y devuelve un número negativo si la fecha representada por el argumento `unaFecha` es anterior a la fecha representada por `otraFecha`.

```
public class Controlador {

```

```
    public void escribirChatEnFichero(String nombreChat, String fName, String
fechaInicial, String fechaFinal) throws IOException, NoCanalException {

        CanalChat canal = this.canales.get(nombreCanal);
         if (canal == null)
              throw new NoCanalException();


         FileOutputStream fos = new FileOutputStream (fName);
         PrintWriter pw = new PrintWriter (fos);

         List<Mensaje> mensajes = canal.getChatPublico().getMensajes();

         for (Mensaje m : mensajes) {
            if (FechaUtils.compareFechas(m.getFechaEmision(), fechaInicial) >= 0
&& FechaUtils.compareFechas(m.getFechaEmision(), fechaFinal) <= 0) {
                pw.println(m.toString());
             }
         }

         pw.close();

    }
}
```