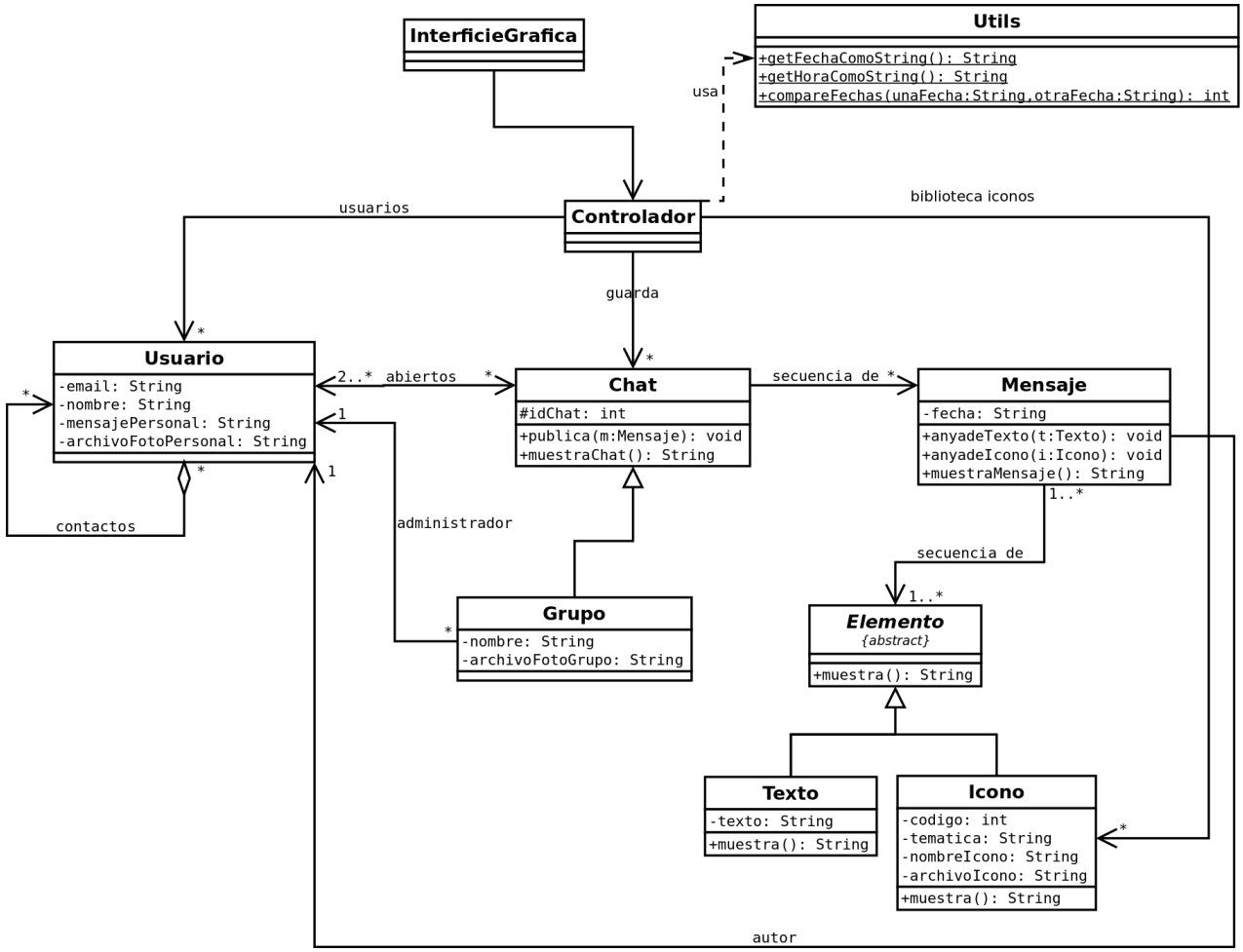
Curso 2015-2016. Cuatrimestre de Primavera. 8 de Julio de 2016
Programación Orientada a Objetos (POO). Examen de Reevaluación.
Duración del examen: 2 h 30 min.

El diagrama de clases UML (incompleto en términos de métodos de las clases) que se muestra en la figura que sigue pretende ser el correspondiente al núcleo de una pequeña aplicación de mensajería instantánea. Cada usuario dispone, en su dispositivo (sea ordenador de sobremesa, portátil, tableta o teléfono móvil inteligente) de un objeto instancia de la clase InterficieGrafica que utiliza para conectarse al servicio.



La aplicación guardará la información de cada usuario en un objeto de la clase Usuario. Para cada usuario la aplicación también guardará sus contactos. A cada usuario se le identifica por su correo electrónico, que actúa como identificador único.

Un usuario puede iniciar conversaciones (objetos de clase Chat) con otros usuarios. Los chats normales lo son entre dos personas. Cada chat tiene un identificador único. También es posible crear grupos, con lo que los chats pasan a ser entre tantas personas como usuarios haya en el grupo. Cada grupo tiene su administrador, que es un usuario del servicio, el nombre del grupo y un nombre de archivo en el que se guarda una imagen correspondiente al grupo.

Los usuarios publican mensajes en los chats ordenados según su instante de creación. Un mensaje incluye la fecha (día y hora) de su publicación, información del usuario autor de éste y una secuencia de elementos constitutivos. Estos elementos pueden ser secuencias de caracteres (strings) o iconos. Cada icono dispone, entre otros atributos, de un valor entero que actúa como código identificador único.

PREGUNTA 1 (1 PUNTO) A partir del diagrama de clases de la figura, 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 mostradas en el diagrama. **NO debéis añadir nada más en el código de estas clases** en vuestras respuestas a esta pregunta.

```
public class InteficieGrafica {
   private Controlador controlador;
}

public class Controlador {
  private Map<String, Usuario> usuarios;   //Clave: email
  private Map<Integer, Chat> chats;    //Clave: idChat
  private Map<Integer, Icono> bibliotecaIconos;    //Clave: código icono
}

public class Usuario {
  private String email, nombre, mensajePersonal, archivoFotoPersonal;
  private Map<String, Usuario> contactos;    //Clave: email
  private Map<Integer, Chat> chatsAbiertos;    //Clave: idChat
}

public class Chat {
  protected int idChat;
  protected Map<String,Usuario> usuarios;    //Clave: email
  protected List<Mensaje> mensajes;
}

public class Grupo extends Chat {
  private String nombre, archivoFotoGrupo;
  private Usuario administrador;
}

public class Mensaje {
  private String fecha;
  private Usuario autor;
  private List<Elemento> elementos;
}

public abstract class Elemento {}

public class Texto extends Elemento {
  private String texto;
}

public class Icono extends Elemento {
```

```
  private int codigo;
  private String tematica, nombreIcono, archivoIcono;
}
```

PREGUNTA 2 (1 PUNTO) Proponed y diseñad un constructor para la clase `Chat`. Proponed y diseñad también un constructor para la clase `Grupo`.

```
public class Chat {
  public Chat (int idChat) {
    this.idChat = idChat;
    this.mensajes = new ArrayList<Mensaje>();
    this.usuarios = new HashMap<String,Usuario>();
  }
}

public class Grupo extends Chat {
  public Grupo (int idChat, String nombre, String archivoFotoGrupo, Usuario
administrador) {
    super (idChat);
    this.nombre = nombre;
    this.archivoFotoGrupo = archivoFotoGrupo;
    this.administrador = administrador;
  }
}
```

PREGUNTA 3 (1,5 PUNTOS) Implementad, en la clase `Controlador` el siguiente método:

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

Este método genera una lista con los usuarios que tienen el máximo número de contactos.

```
public class Controlador {
  public List<Usuario> getUsuariosConMaximoNumeroDeContactos(){
    ArrayList<Usuario> usuariosMaxContactos = new ArrayList<Usuario>();
    int maxContactos = 0;
    for (Usuario user : this.usuarios.values()) {
      if (user.getNumContactos() > maxContactos)
        maxContactos = user.getNumContactos();
    }

    if (maxContactos > 0)
    {
      for (Usuario user : this.usuarios.values()) {
        if (user.getNumContactos() == maxContactos)
          usuariosMaxContactos.add(user);
      }
```

```
    }

    return usuariosMaxContactos;  //Lista vacía si nadie tiene contactos…
  }
}

//Método utilizado como soporte:
public class Usuario {
  public int getNumContactos() {
    return this.contactos.size();
  }
}
```

PREGUNTA 4 (1 PUNTO) Implementad, en la clase `Mensaje`, el siguiente método:

```
public int numIconos() ;
```

Este método devuelve el número de iconos que aparecen en el mensaje en cuestión.

Suponed para ello que la clase Elemento dispone del método `isIcono()`. El método `isIcono()` de Texto devuelve false. El método `isIcono()` de Icono devuelve true.

```
public class Mensaje {
  public int numIconos() {
    int numIconos = 0;
    for (Elemento elem : this.elementos) {
      if (elem.isIcono())
        numIconos++;
    }
    return numIconos;
  }
}
```

PREGUNTA 5 (1,5 PUNTOS) Implementad, en la clase `Controlador`, el siguiente método:

```
public int cuentaNumeroDeIconosEnviadosPorUsuario(String mailUsuario) throws
NoUsuarioException;
```

Este método cuenta el número de iconos que ha enviado a todos los chats en los que participa el usuario cuya dirección de correo electrónico sea el valor pasado por el argumento `mailUsuario`. Si no hay ningún usuario con la dirección de correo electrónico antes mencionada, el método lanzará una excepción `NoUsuarioException`.

```
public class Controlador {
  public int cuentaNumeroDeIconosEnviadosPorUsuario(String mailUsuario) throws
NoUsuarioException {
    Usuario user = this.usuarios.get(mailUsuario);
```

```
    if (user == null){
      throw new NoUsuarioException();
    }

    int numIconos = 0;
    Map<Integer,Chat> chatsAbiertos = user.getChatsAbiertos();
    for (Chat chat : chatsAbiertos.values()) {
      List<Mensaje> mensajes = chat.getMesajes();
      for (Mensaje mensaje : mensajes) {
        if (mensaje.getAutor().getEmail().equals(mailUsuario)
          numIconos += mensaje.numIconos();
      }
    }
    return numIconos;
  }
}
```

PREGUNTA 6 (1,5 PUNTOS) Suponed que la clase Utils incorpora el método:

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

que devuelve un entero positivo si la fecha representada por el argumento `unaFecha` es una fecha posterior a la fecha representada por el argumento `otraFecha`, un 0 si ambos argumentos representan a la misma fecha, y un entero negativo si la fecha representada por el argumento `unaFecha` es una fecha anterior a la fecha representada por el argumento `otraFecha`

Implementad el método siguiente de la clase Controlador:

```
public  List<Mensaje> getMensChatEntreFechas(String mailUsuario, String idChat,
String fechaInicio, String fechaFin)  throws NoUsuarioException,
NoChatException;
```

Este método debe generar una lista que incluya aquellos mensajes del autor con la dirección de correo electrónico indicada en el argumento `mailUsuario`, que hayan sido enviados entre la fecha indicada en el argumento `fechaInicio` y la fecha indicada en el argumento `fechaFin` (ambas incluidas) al chat identificado por el valor del argumento `idChat`. Si no hay ningún usuario con la dirección de correo electrónico antes mencionada, el método lanzará una excepción `NoUsuarioException`. Si no hay ningún chat con un identificador igual al valor del argumento `idChat` el método lanzará una excepción `NoChatException`.

```
public class Controlador {
  public  List<Mensaje> getMensChatEntreFechas(String mailUsuario, String
idChat, String fechaInicio, String fechaFin)  throws NoUsuarioException,
NoChatException {
    Usuario user = this.usuarios.get(mailUsuario);
    Chat chat = this.chats.get(idChat);
    if (user == null) {
      throw new NoUsuarioException();
```

```
    }
    if (chat == null) {
      throw new NoChatException();
    }

    List<Mensaje> entreFechas = new ArrayList<Mensaje>();
    List<Mensaje> mensajes = chat.getMensajes();
    for (Mensaje mensaje : mensajes) {
      if(mensaje.getAutor().getEmail().equals(mailUsuario) &&
       Utils.compareFechas(mensaje.getFecha(), fechaInicio) >= 0 &&
       Utils.compareFechas(mensaje.getFecha(), fechaFin) <= 0)
         entreFechas.add(mensaje);
      }
    }
    return entreFechas;
  }
}
```

PREGUNTA 7 (1 PUNTO) Implementad en la clase Mensaje el método:

`public String muestraMensaje();`

Este método genera un String resultante de concatenar los siguientes componentes:

- El string "<autor>", la dirección de correo electrónico del usuario que ha enviado el mensaje y el string "</autor>", seguido de
- El string "<fecha>", la fecha de publicación del mensaje y el string "<\fecha>", seguido de
- El string resultante de concatenar los strings devueltos por los métodos `muestra()` de cada uno de los elementos que lo forman, y encapsula el resultado entre la cabecera "<mensaje>" y el pie "</mensaje>".

Para construir el código de este método debéis construir también el código del método

`public String muestra();`

de las clases Texto e Icono. El método muestra() de Texto debe devolver el valor del atributo texto encapsulado entre la cabecera "<texto>" y el pie "</texto>" . El método de Icono debe devolver el valor del atributo código encapsulado entre la cabecera "<icono>" y el pie "</icono>".

Mostrad también cómo debe aparecer el método muestra() en la clase Elemento.

```
public class Elemento {
   public abstract String muestra();
}

public class Texto extends Elemento {
  public String muestra() {
    return "<texto>" + this.texto + "</texto>";
```

```
  }
}

public class Icono extends Elemento {
  public String muestra() {
    return "<icono>" + this.codigo + "</icono>";
  }
}

public class Mensaje {
  public String muestraMensaje() {
    String m = "<autor>" + this.autor.getEmail() + "</autor>";
    m += "<fecha>" + this.fecha + "</fecha>";
    m += "<mensaje>";
    for (Elemento elemento : this.elementos) {
      m += elemento.muestra();
    }
    m += "</mensaje>";
    return m;
  }
}
```

PREGUNTA 8 (1,5 PUNTOS) Implementad, en la clase `Controlador`, el siguiente método:

```
public void guardaMensajesChat(String idChat, String nombreFicheroCompleto)
throws NoChatException;
```

Este método intenta guardar en un archivo cuyo nombre completo (es decir, el que incluye el nombre completo del directorio en el que está y el nombre local del fichero; en otras palabras, su "pathName") se pasa a través del argumento `nombreFicheroCompleto` todos los mensajes enviados al chat cuyo identificador coincide con el valor del argumento `idChat`. Si no hay ningún chat con un identificador igual al valor del argumento `idChat` el método lanzará una excepción `NoChatException`. Cada mensaje aparecerá separado del siguiente por un final de línea. El contenido de cada mensaje será el resultado de invocar al método `muestraMensaje()` de la clase Mensaje. Notad que este método no lanza ninguna excepción generada por las operaciones de entrada/salida.

```
public class Controlador {
  public void guardaMensajesChat(String idChat, String nombreFicheroCompleto)
throws NoChatException {
    Chat chat = this.chats.get(idChat);
    if (chat == null) {
      throw new NoChatException();
    }

    try {
       FileOutputStream fos = new FileOutputStream(nombreFicheroCompleto,
false);
```

```
        PrintWriter pw = new PrintWriter(fos);

        List<Mensaje> mensajes = chat.getMensajes();
        for (Mensaje mensaje : mensajes) {
          pw.println(mensaje.muestraMensaje());  //Añade "\n" al final de línea
        }

        pw.close();
      }
      catch (IOException e) {}
/* Dentro del catch podrían haber instrucciones que presenten mensajes por
pantalla */
   }
}
```