ANDREA FERNÁNDEZ JOGLAR – 1º DAM

TAREA DE ENTREGA

ÍNDICE

1. INTRODUCCIÓN

Los organizadores del festival de música y danza *Raíces*, con más de cincuenta ediciones a sus espaldas, quieren almacenar la información relativa a los artistas que van a participar en la próxima edición de este evento celebrado cada verano. Aunque ya tienen algunos grupos confirmados, necesitan diseñar un sistema de almacenamiento flexible, que les permita inscribir nuevos artistas como participantes una vez que puedan confirmar su asistencia a través de sus mánagers.

El fundador de *Raíces*, el famoso productor teatral Guillerme Nosti, recientemente fallecido a los 98 años, llevaba casi seis décadas organizando el festival y era un entusiasta de los espectáculos musicales y de danza. Como homenaje, la nueva organización ha decidido mantener la tradicional división de artistas participantes entre grupos de música y grupos de danza.

El pasado mes de enero, se encontraron entre los papeles de la oficina de Guillerme algunos documentos y bocetos con indicaciones sobre los datos que deberían almacenarse en relación con los artistas que acudan al festival; un hecho fortuito que ha sido tomado como un pilar fundamental para el planteamiento del diseño del evento. Aquellos manuscritos de Nosti, usados para organizar la primera edición del festival en el año 1968, recogían algunas recomendaciones que se reproducen a continuación de manera parcial:

Consejos para organizar el Festival Raíces (texto manuscrito de Guillerme Nosti)

Enero de 1968,

1. *Es importante ser meticulosos con el almacenamiento de la información relativa a los artistas. Resulta imprescindible guardar el* ***nombre****, los* ***emolumentos*** *del grupo (es decir, lo que modernamente se llama* ***caché)*** *para poder saber si podemos cubrir los gastos con la subvención anual y l****os datos personales del mánager*** *o productor.*
2. *Ser cuidadosos con la información también supone generar identificadores claros. Como siempre digo, conviene tener* ***una parte del identificador común a todos los artistas*** *y* ***otra parte que los identifique como grupos musicales o grupos de danza****. Ojalá inventaran pronto una máquina de escribir que* ***actualizara automáticamente los identificadores*** *sin necesidad de tener que ir poniéndolos uno a uno…*
3. *Los* ***mánagers****… ¡Ay, los malditos mánagers! A veces tienen más ego ellos que sus representados. Para no tener problemas, vamos a guardar todos los datos posibles para que no puedan escaquearse cuando contactemos con ellos, esto es: el* ***nombre****, los* ***apellidos****, el* ***número de teléfono*** *–¿algún día inventarán un* ***servicio instantáneo para enviar y recibir correo****?– y la* ***empresa*** *a la que pertenecen.*
4. *Por supuesto, cuando referenciemos los datos de los artistas, debemos incluir entre ellos a los mánagers (¡no vaya a ser que se pongan celosos!).*
5. *Sobre los* ***grupos musicales****, tenemos que reflejar su* ***estilo musical*** *y sus requisitos* ***técnicos****. Me gustaría crear un festival plural, atractivo para diversos públicos.*
6. *Sobre los grupos de danza, queremos traer al festival distintos* ***tipos de bailes,*** *con compañías de* ***diversas nacionalidades****. Aspiramos a la creación de un evento de dimensión internacional.*
7. *Al almacenar todos los datos de los artistas que participen en el festival tenemos que ser eficientes. Hay que ser capaces de generar un sistema de guardado que permita* ***insertar*** *nuevos artistas,* ***buscarlos****,* ***borrarlos*** *en caso de que surjan imprevistos y* ***consultar toda la información*** *de manera eficiente.*

A partir de estas recomendaciones encontradas, los organizadores han decidido actualizar los propósitos de Nosti con las posibilidades que ofrecen los tiempos actuales. De este modo, diseñaron una aplicación capaz de:

- Mostrar mediante un menú todas las operaciones de utilidad que se pueden realizar sobre los datos:
  - Cargar los datos de los artistas ya confirmados

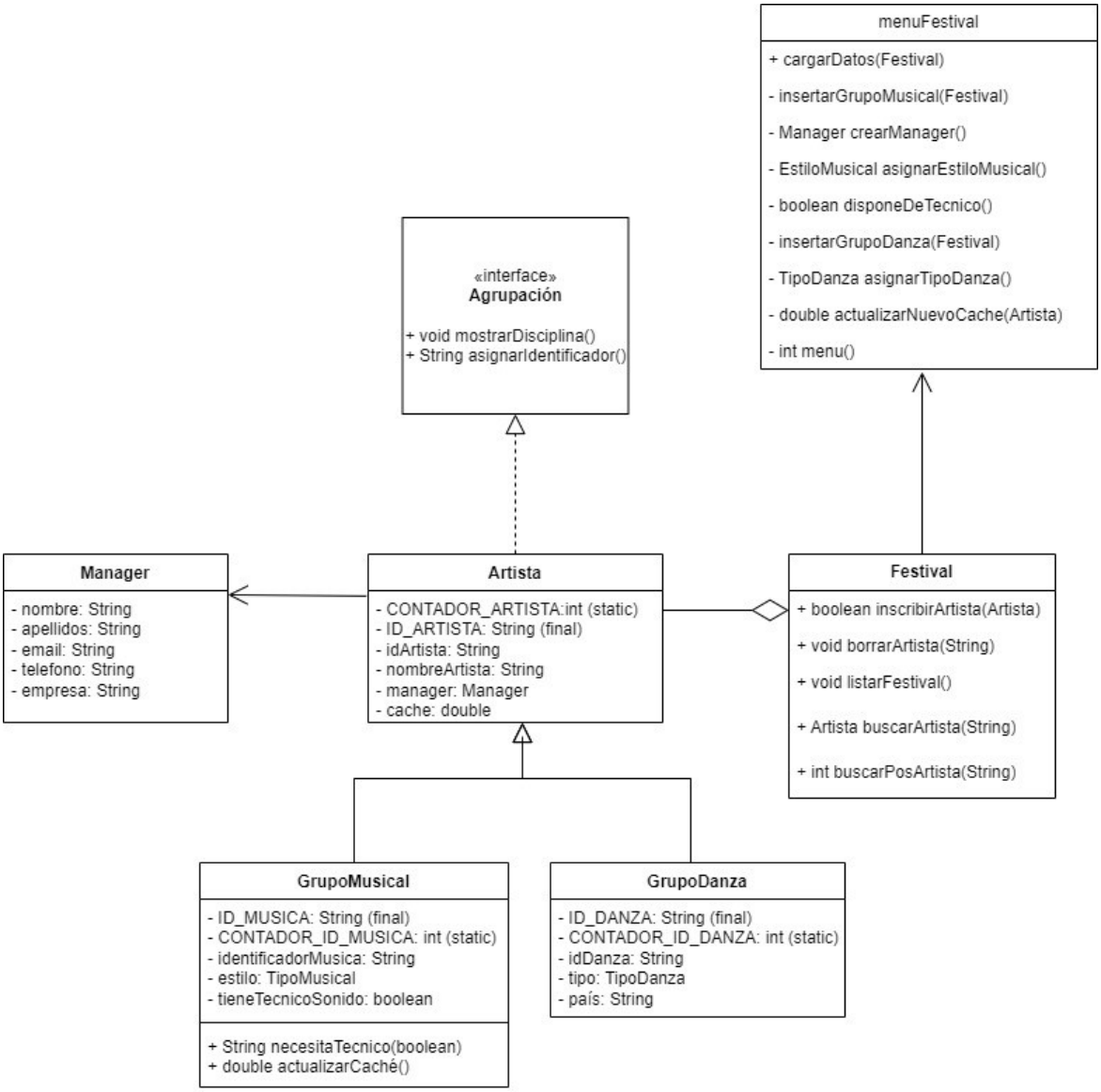
- Insertar grupos musicales en el festival, con la posibilidad de:
  - introducir los datos relativos al mánager
  - especificar el estilo musical. En esta edición se contratarán grupos de música pop, rock, punk y metal.
- Insertar grupos de danza en el festival, con la posibilidad de
  - especificar el país de origen del grupo, respetando las aspiraciones de repercusión internacional del fundador
  - introducir el estilo de danza que realizan. En esta edición, se contratarán grupos de danza clásica, tradicional, contemporánea y urbana.
- Actualizar el caché en caso de que sea necesario.
- Listar todos los artistas participantes en el festival.
- Buscar artistas a partir de un identificador creado previamente y que va incrementando según se añaden nuevos artistas.
- Borrar artistas en caso de que se caigan del cartel del festival.

Cumpliendo con la voluntad de Guillermo Nosti, un auténtico visionario de su tiempo, la organización decide añadir un **email** como forma de contacto del festival con el mánager de los artistas, respetando en su práctica totalidad las indicaciones anteriormente especificadas.

Con el objetivo de clarificar toda la información, se ha decidido diseñar un esquema de diseño previo a la implementación de la aplicación, a modo de **diagrama de clases**, que se representa en la página siguiente.

Además, se añadieron al final unas especificaciones que permiten **justificar las decisiones** que se tomaron a la hora de implementar el proyecto e introducir mejoras al primigenio plan de diseño elaborado por Nosti.

2. DIAGRAMA DE CLASES



3. JUSTIFICACIÓN DE LAS DECISIONES TOMADAS EN EL PROYECTO

Para la correcta implementación del proyecto se tomaron ciertas decisiones tomando como punto de partida el diagrama de clases anteriormente expuesto.

1. En primer lugar, se decide crear una interface llamada **«Agrupación»** que no tiene atributos. En esta interface, se plantean dos métodos que serán utilizados en las clases que dependen de ella:
   - El primero de ellos, asignarIdentificador ( ), será utilizado tanto por la clase genérica Artista como por sus dos subclases GrupoMusical y GrupoDanza. Este método permite asignar un identificador único a cada una de estas categoría.
   - El segundo, mostrarDisciplina ( ), va a implementarse en las subclases GrupoMusical y GrupoDanza. Va a permitir especificar la disciplina artística a la que pertenece cada artista del festival
2. A continuación, se crea una clase **Artista** que tiene las siguientes especificaciones:
   - un valor constante ID_ARTISTA, que se inicializa como "ART-"
   - una variable static contadorArtista, inicializada en 0 y que se va autoincrementando conforme se crean nuevos artistas
   - un atributo idArtista de tipo String, que indica el identificador único de cada artista
   - un atributo nombreArtista de tipo String, en el que se indica el nombre de cada artista
   - un atributo manager de la clase Manager, que se especificará más adelante, que recoge toda la información relativa al representante del artista.
   - un atributo caché, de tipo real, que indica el dinero que cobra cada grupo por actuación

   Todos los atributos serán privados, por lo que la clase incluirá los métodos get y set correspondientes para que sean accesible, el método toString( ) y un constructor con todos los parámetros, menos el idArtista, que se creará a partir del método asignarIdentificador( ) implementado desde la interface Agrupación.

   El método asignarIdentificador( ) genera un identificador que combina el valor constante del ID_ARTISTA y el valor del contador static.

3. La clase **Manager** se relaciona mediante asociación con la clase Artista, ya que se utiliza como atributo en esta última. Un manager deberá tener como atributos, todos ellos de tipo String, el nombre, apellidos, teléfono, email y empresa de management.

   Incluirá el constructor con todos los parámetros, los métodos get y set y el método toString.

4. La clase Artista cuenta con dos subclases: **GrupoMusical y GrupoDanza**.

   a) **GrupoMusical**: al igual que su superclase, cuenta con una constante ID_MUSICA inicializada como "GM", un contador autoincrementado de identificadores del grupo y los siguientes atributos privados:

   - un idMusica de tipo String, correspondiente con el identificador generado a partir del método asignarIdentificador( )
   - un atributo que indique el tipo de estilo musical a partir de un enumerado que solo podrá tomar los valores pop, rock, punk y metal
   - un booleano que determine si el grupo tiene un técnico de sonido propio o no.

   Cuenta con un constructor que recibe todos los parámetros a excepción del idMusica, que se inicializa a partir del método asignarIdentificador( ) que combina la constante ID_MUSICA y el contador correspondiente; así como con los métodos get y set.

   En caso de que el grupo no tenga un técnico propio, el festival deberá tener en cuenta que el caché del grupo se incrementará 500 € para cubrir los gastos derivados de la contratación de un técnico de sonido a cuenta del festival. Se creará un método para actualizar el caché que retorne el nuevo valor.

   Se decide crear un método que convierta el booleano del técnico de sonido en un String para poder mostrarlo por pantalla como "SÍ" o "NO".

   Se implementa el método mostrarDisciplina( ) de la interface en el que se muestre por pantalla todos los datos relevantes del grupo musical.

   b) **GrupoDanza**: cuenta con una constante ID_DANZA inicializada como "GD", un contador autoincrementado de identificadores del grupo y los siguientes atributos privados:

   - un idDanza de tipo String, correspondiente con el identificador generado a partir del método asignarIdentificador( )

- un atributo que indique el tipo de estilo de danza a partir de un enumerado que solo podrá tomar los valores clásica, tradicional, contemporánea y urbana
- un atributo String que indique el país de origen de la agrupación.

Tendrá un constructor con todos los parámetros, excepto el idDanza, creado a partir del método heredado de asignación de identificadores, combinando la constante ID_DANZA con el contador que corresponda; así como los métodos get y set.

Se implementa el método mostrarDisciplina( ) de la interface en el que se muestre por pantalla todos los datos relevantes del grupo de danza.

5. Deberá crearse una clase **Festival** en la que se puedan incluir métodos que permitan insertar artistas, borrarlos y buscarlos a partir de su identificador único, así como mostrar por pantalla todos los artistas que componen el cartel del festival. Se incluirá también un método auxiliar para buscar artistas a partir de su identificador, que devuelva la posición en la que se encuentra.
6. Por último, se incluye una clase que actúe como menú del festival, a partir del que se puedan probar todas las operaciones de consulta y modificación sobre los datos: insertar grupo musical, insertar grupo de danza, mostrar los artistas por pantalla, buscar artista, eliminarlo y modificar caché.

   Se incluyen igualmente todos los métodos auxiliares que permitan rellenar los datos para la inscripción de cada artista con éxito.

4. CLASES DE EQUIVALENCIA

En este apartado recogemos las pruebas unitarias realizadas con JUnit para comprobar los principales métodos del programa ideado para el festival *Raíces*. En primer lugar, mostraremos el método que se va a comprobar; seguido de las tablas relativas a las clases de equivalencia, y, por último, la tabla con los casos de prueba.

1. boolean inscribirArtista(Artista)

```java
/**
 * Metodo para inscribir artistas en el festival
 * @param a Artista
 * @return boolean
 */
public boolean inscribirArtista (Artista a) {
    boolean resultado = false;

    if (artistas == null) {
        artistas = new Artista [1];
        artistas[0] = a;
        resultado = true;
    }
    else {
        if (buscarArtista(a.getIdArtista()) != null) {
            resultado = false;
        }
        else {
            int nuevoTamanio = artistas.length + 1;
            Artista [] nuevoArtistas = new Artista [nuevoTamanio];

            for (int i = 0; i < artistas.length; i++) {
                nuevoArtistas[i] = artistas[i];
            }
            int pos = nuevoTamanio - 1;
            nuevoArtistas[pos] = a;
            artistas = nuevoArtistas;
            resultado = true;
        }
    }
    return resultado;
}
```

CLASES DE EQUIVALENCIA

Condiciones de entrada	Clase equivalencia válida	Clase equivalencia no válida
Creación Festival inicial vacío	NumArtistas = 0 (1)	NumArtistas != 0 (2)
Tipo de dato	Parámetro = Artista (3)	Parámetro != Artista (4)
inscribir según resultado	Se inscriben artistas (5)	No se inscriben artistas (6)
cambio longitud del vector	Aumenta la longitud (7)	No cambia la longitud (8)

CASOS DE PRUEBA

Condición	Clase equivalencia	Resultado
Festival festival = new Festival()	1	0
Festival festival = new Festival()	2	ERROR
Artista	3, 5	true
int	4, 6	ERROR
inscribir 1 Artista	3, 5, 7	numArtistas = +1
Intentar inscribir 1 artista no válido	8	numArtistas no cambia
Después de inscribir 6 Artista	3,5,7	numArtistas = + 6
Después de inscribir 6 artistas no válidos	8	numArtistas no cambia

2. boolean borrarArtista (String)

```java
/**
 * Metodo para borrar un artista previamente inscrito en el festival
 * @param id String
 * @return boolean
 */
public boolean borrarArtista (String id) {
    boolean encontrado = false;

    int posicion = buscarPosArtista(id);
    if (posicion == -1) {
        encontrado = false;
    }
    else {
        if(posicion != -1) {
            int nuevoTamanio = artistas.length - 1;

            Artista [] vNuevo = new Artista[nuevoTamanio];

            for (int i = 0; i < posicion; i++) {
                vNuevo[i] = artistas[i];
            }

            for (int i = posicion + 1; i < artistas.length; i++) {
                vNuevo[i-1] = artistas[i];
            }

            artistas = vNuevo;
            encontrado = true;
        }
    }

    return encontrado;
}
```

CLASES DE EQUIVALENCIA

Condiciones de entrada	Clase equivalencia válida	Clase equivalencia no válida
Tipo de parámetro	String id almacenado(1)	String id no almacenado (2)
borrar según resultado	Se borra el artista (3)	No se borra el artista (4)
cambio longitud del vector	Disminuye la longitud (5)	Longitud no varía (6)

CASOS DE PRUEBA

Condición	Clase equivalencia	Resultado
String id1 (almacenado)	1	true
String id no almacenado / cadena vacía	2	false
borrar 1 artista (id almacenado)	1, 3, 5	numArtistas = -1
Intentar borrar 1 id no almacenado	2, 4, 6	numArtistas (no varía)
Después de borrar 3 id inscritas previamente	3, 5	numArtistas -3
Después de borrar 3 id no válidas	4, 6	numArtistas (no varía)

3. Artista buscarArtista (String)

```
/**
 * Metodo para buscar artistas dentro del festival a partir de su identificador
 * @param id String
 * @return Artista
 */
public Artista buscarArtista (String id) {
    Artista art1 = null;
    for (int i = 0; i < artistas.length; i++) {
        Artista a = artistas[i];
        if (a.getIdArtista().compareTo(id) == 0) {
            art1 = a;
        }
    }
    return art1;
}
```

CLASES DE EQUIVALENCIA

Condiciones de entrada	Clase equivalencia válida	Clase equivalencia no válida
Tipo de parámetro	String id almacenado(1)	String id no almacenado (2)
Buscar según resultado	Busca el artista (3)	No busca el artista (4)

CASOS DE PRUEBA

Condición	Clase equivalencia	Resultado
String id (almacenado)	1	true
String no almacenado / cadena vacía	2	false
Buscar artista (id almacenado)	1, 3	true
Buscar artista (id no almacenado)	2, 4,	false

4. double actualizarNuevoCache (double, Artista)

```
/**
 * Permite actualizar el cache del artista
 * @param a Artista
 * @return real
 */
public double actualizarNuevoCache(double cache, Artista a) {
    a.setCache(cache);
    return a.getCache();
}
```

CLASES DE EQUIVALENCIA

Condiciones de entrada	Clase equivalencia válida	Clase equivalencia no válida
Tipo de parámetro	double, Artista(1)	No double o no Artista (2)
Buscar según resultado	Actualiza el caché (3)	No lo actualiza (4)

CASOS DE PRUEBA

Condición	Clase equivalencia	Resultado
Actualiza 13000, gMusical1	1, 3	true
String, gMusical1	2	ERROR
No actualiza 5000, gMusical1	4	false