# Práctica 4: Herencia, Interfaces y Excepciones

**Inicio:** Semana del 20 de marzo.
**Duración:** 3 semanas.
**Entrega:** En Moodle, una hora antes del comienzo de la siguiente práctica según grupos (semana del 17 de Abril)
**Peso de la práctica: 30%**

En esta práctica se pide diseñar e implementar una aplicación para gestionar listas de música, donde los usuarios pueden votar sus canciones y álbumes favoritos, y recibir recomendaciones de nuevas canciones y álbumes basadas en las preferencias de otros usuarios con gustos similares, o en la música más popular entre el resto de usuarios.

El objetivo de la práctica es aprender a utilizar técnicas de orientación a objetos más avanzadas que en las prácticas anteriores. En concreto, se hará énfasis en los siguientes conceptos:

- *interfaces*
- *herencia*
- *excepciones*
- *colecciones*

## Apartado 1. Fonoteca básica (2 puntos)

Comenzamos creando la clase `Fonoteca`, que debe permitir gestionar álbumes y listas de música con las siguientes características:

- Cada *canción* tiene un título y una duración especificada en minutos y segundos. Si se intenta crear una canción con una duración inválida – minutos o segundos negativos, o más de 60 segundos – se ajustan a 0 o a 60.
- Los *álbumes de música* tienen un título, un artista, una o más canciones, y pueden tener asignado o no un estilo musical entre los siguientes: rock, pop, disco, metal, reggaetón, jazz, soul y clásica. Al crear un álbum, se debe indicar la lista de canciones que contiene, y posteriormente, no debe ser posible modificar dicha lista. La duración de un álbum es la suma de las duraciones de las canciones que contiene.
- Las *listas de música* tienen un título, y pueden incluir canciones, álbumes y otras listas de música existentes en la fonoteca. Al contrario que los álbumes, siempre es posible modificar el contenido de una lista para añadir nuevos elementos. La duración de una lista de música es la suma de la duración de las canciones que contiene, ya sea directamente, o indirectamente en sus álbumes y sublistas.

A continuación, tienes un programa de prueba con la salida esperada. Como puedes observar, tanto álbumes como listas de música se crean a través de un objeto `Fonoteca`. Esto nos servirá para poder controlar errores en el próximo apartado. El método `crearAlbum` recibe un número variable de canciones, y devuelve `null` si se pasan 0 canciones.

```
public class FonotecaTester {
  public static void main(String[] args) {
        FonotecaTester main = new FonotecaTester();
        Fonoteca fonoteca = main.crearMusica();
        fonoteca.mostrar();
  }

  public Fonoteca crearMusica() {
        Cancion[] canciones ={
                    new Cancion("Radio ga ga", 5, 48), // Canción Radio ga ga, con duración 5:48
                    new Cancion("Tear it up", 3, 28),
                    new Cancion("It's a hard life", 4, 8),
                    new Cancion("Resistire", 4, 4),
                    new Cancion("Dos corazones", 3, 6)};

        Fonoteca fonoteca = new Fonoteca();

        Album album1 = fonoteca.crearAlbum("The Works", "Queen", EstiloMusical.ROCK,
                                           canciones[0], canciones[1], canciones[2]);
        fonoteca.crearAlbum("Resistire", "Duo dinamico", canciones[3], canciones[4]); //sin estilo musical

        ListaMusica favoritas = fonoteca.crearListaMusica("Mis favoritas");
        fonoteca.aniadirMusicaALista(favoritas, album1)
                .aniadirMusicaALista(favoritas, canciones[3])
                .aniadirMusicaALista(favoritas, canciones[4]);

        System.out.println(fonoteca.crearAlbum("Las 4 estaciones", "Vivaldi", EstiloMusical.CLASICA));

        return fonoteca;
  }
}
```

### Salida esperada:

```
null
ALBUM: The Works, ARTISTA: Queen, DURACION: 13:24, ESTILO: ROCK
   1.Radio ga ga (5:48)
   2.Tear it up (3:28)
   3.It's a hard life (4:08)
--------------
ALBUM: Resistire, ARTISTA: Duo dinamico, DURACION: 7:10, ESTILO: SIN ESTILO
   1.Resistire (4:04)
   2.Dos corazones (3:06)
--------------
LISTA: Mis favoritas, DURACION: 20:34
   1.ALBUM: The Works, ARTISTA: Queen, DURACION: 13:24, ESTILO: ROCK
      1.Radio ga ga (5:48)
      2.Tear it up (3:28)
      3.It's a hard life (4:08)
   2.Resistire (4:04)
   3.Dos corazones (3:06)
--------------
```

### Notas adicionales:

- Organiza tu código, así como los testers, en paquetes. No olvides crear programas de prueba adicionales al del enunciado.
- Restringe la creación de álbumes y listas para que se haga llamando a los métodos de `Fonoteca`, y no sea posible llamar a los constructores de álbumes y listas desde fuera del paquete.

## Apartado 2. Control de errores mediante excepciones (2 puntos)

En este apartado, extenderemos la fonoteca para realizar un control básico de errores mediante excepciones.

Si a una lista se intenta añadir una canción, álbum o lista que no pertenece a la fonoteca, se lanzará la excepción `ExcepcionMusicaNoExistente`. Además, los álbumes y las listas no pueden contener canciones repetidas directa ni indirectamente (por ejemplo, no podrían contener la canción "*Resistiré*", y una lista con la canción "*Resistiré*"). Si se detecta esta situación al añadir una nueva canción, álbum o lista, se lanzará la excepción `ExcepcionCancionRepetida`. Esta excepción almacenará la canción que se repite, para que luego pueda mostrarse. Dos canciones se consideran iguales si tienen el mismo nombre y duración.

El siguiente tester ilustra el uso de estas excepciones. Como puedes ver, no se puede crear el álbum "*Resistiré*" ya que tiene una canción repetida (pese a que los objetos canción ocupan posiciones distintas en memoria), y no se puede añadir la canción "*Tear it up*" a la lista porque ya está en el álbum "*The Works*". Además, ambas excepciones son compatibles con `ExcepcionFonoteca`, por lo que debes realizar un diseño que evite duplicidades en las excepciones.

```
public class FonotecaTesterErrores {
  protected Album album1; // protected porque extenderemos esta clase en los próximos apartados
  protected Cancion[] canciones ={ new Cancion("Radio ga ga", 5, 48),
                                   new Cancion("Tear it up", 3, 28),
                                   new Cancion("It's a hard life", 4, 8),
                                   new Cancion("Resistire", 4, 4),
                                   new Cancion("Dos corazones", 3, 6)};

  public static void main(String[] args) {
        FonotecaTesterErrores main = new FonotecaTesterErrores();
        Fonoteca fonoteca = new Fonoteca();
        main.crearMusica(fonoteca);
        fonoteca.mostrar();
  }

  public void crearMusica(Fonoteca fonoteca) {
        try {
              this.album1 = fonoteca.crearAlbum("The Works", "Queen", EstiloMusical.ROCK,
                                                canciones[0], canciones[1], canciones[2]);
              fonoteca.crearAlbum("Resistire", "Duo dinamico",
                                  canciones[3], new Cancion("Resistire", 4, 4)); // cancion repetida
        } catch (ExcepcionCancionRepetida cr) {
              System.err.println(cr);
        }

        ListaMusica favoritas = fonoteca.crearListaMusica("Mis favoritas");
        try {
              fonoteca.aniadirMusicaALista(favoritas, album1)
                      .aniadirMusicaALista(favoritas, canciones[1]);
        } catch (ExcepcionFonoteca e) {
              System.err.println(e);
        }
  }
}
```

## Salida esperada:

```
music.exceptions.ExcepcionCancionRepetida: La cancion Resistire esta repetida
music.exceptions.ExcepcionCancionRepetida: La cancion Tear it up esta repetida
ALBUM: The Works, ARTISTA: Queen, DURACION: 13:24, ESTILO: ROCK
   1.Radio ga ga (5:48)
   2.Tear it up (3:28)
   3.It's a hard life (4:08)
--------------
LISTA: Mis favoritas, DURACION: 13:24
   1.ALBUM: The Works, ARTISTA: Queen, DURACION: 13:24, ESTILO: ROCK
      1.Radio ga ga (5:48)
      2.Tear it up (3:28)
      3.It's a hard life (4:08)
--------------
```

## Notas adicionales:

- El tester del apartado 1 dejará de compilar una vez que añadas excepciones. Debes arreglarlo, asegurando que da el mismo resultado.
- Para comprobar que dos objetos son iguales, puedes sobrescribir el método `equals` de `Object`. De esta manera, podrás comprobar no sólo si los objetos apuntan a la misma dirección de memoria (==), sino implementar nociones de igualdad específicas (por ejemplo, que comparen los objetos campo a campo). El método `equals` se usa muy frecuentemente para comparar objetos. Por ejemplo, métodos como `lista.contains(obj)` usan `equals` para comprobar si existe un objeto igual a `obj` en `lista`. Recuerda que si sobrescribes `equals`, también debes sobrescribir el método `hashCode`, de tal manera que dados dos objetos `o1` y `o2`, si `o1.equals(o2)` devuelve true, entonces `o1.hashCode()==o2.hashCode()` debe ser true.

## Apartado 3. Usuarios y valoraciones (2 puntos)

En este apartado, añadiremos un componente para almacenar valoraciones de usuarios a canciones y álbumes. La estrategia de diseño será crear un componente general, que podamos utilizar para almacenar valoraciones de cualquier tipo de usuario y elemento, como por ejemplo películas, libros, o destinos de viaje. Utilizaremos interfaces para realizar este diseño. Esto es adecuado ya que, en el apartado 4, extenderemos el componente para que proporcione recomendaciones de música a los usuarios por distintos criterios.

En primer lugar, la fonoteca debe permitir el registro de nuevos usuarios mediante su nombre y nickname. Este último debe ser único en la fonoteca. Los usuarios registrados pueden valorar las canciones y álbumes existentes en la fonoteca, pero no las listas. Hay dos valoraciones posibles, `LIKE` y `DISLIKE`, para indicar las canciones y álbumes que le gustan o no al usuario, respectivamente. Si un usuario no ha valorado una canción, pero sí el álbum que la contiene, se considerará que la valoración del usuario para esa canción es la misma que la de su álbum. La fonoteca debe permitir listar la música que ha valorado un usuario.

Para almacenar usuarios y sus valoraciones, la fonoteca usará una clase, que debes construir, y que será conforme a la siguiente interfaz:

```
public interface IAlmacenValoraciones {
	boolean		addUsuario(IUsuario usuario);
	boolean		addRecomendable(IRecomendable elemento);
	void		addValoracion(IUsuario usuario, IRecomendable elemento, Valoracion valoracion);
	boolean		haValorado(IUsuario usuario, IRecomendable elemento);
	Collection<IRecomendable> elementosValorados(IUsuario usuario);
	Valoracion	valoracion(IUsuario usuario, IRecomendable elemento);
}
```

Los métodos de esta interfaz deben tener el siguiente comportamiento:

- `addUsuario(IUsuario)`: registra un nuevo usuario en el almacén.
- `addRecomendable(IRecomendable)`: registra un nuevo elemento recomendable en el almacén.
- `addValoracion(IUsuario, IRecomendable, Valoracion)`: registra la valoración de un usuario a un elemento.
- `haValorado(IUsuario, IRecomendable)`: devuelve si un usuario dado ha valorado un elemento.
- `elementosValorados(IUsuario)`: devuelve una colección con los elementos valorados por un usuario.
- `valoracion(IUsuario, IRecomendable)`: devuelve la valoración dada por un usuario a un elemento.

Como puedes ver, la interfaz `IAlmacenValoraciones` usa a su vez la interfaz `IRecomendable` para indicar la clase de objetos a valorar; la interfaz `IUsuario` para los usuarios que realizan las valoraciones; y el enumerado `Valoracion` con los valores posibles `LIKE` y `DISLIKE`. A continuación, tienes la definición de todas ellas.

```
public interface IRecomendable {
   String getDescripcion();
}
public interface IUsuario {
   String getId();          // un ID, que debe ser único
}
public enum Valoracion { LIKE, DISLIKE }
```

Como ejemplo de uso, el siguiente programa debe dar la salida de más abajo

```
public class FonotecaTesterValoraciones extends FonotecaTesterErrores {
  public static void main(String[] args) {
	FonotecaTesterValoraciones main = new FonotecaTesterValoraciones();
	Fonoteca fonoteca = new Fonoteca();
	main.crearMusica(fonoteca);
	main.valoraciones(fonoteca);
  }

  public void valoraciones(Fonoteca fonoteca) {
	Usuario usuario1 = fonoteca.registrarUsuario("Sonia Melero Vegas", "smelero"),
		usuario2 = fonoteca.registrarUsuario("Miguel Cuevas Alonso", "mcuevas");
	fonoteca.valorar(usuario1, this.canciones[0], Valoracion.DISLIKE)
		.valorar(usuario1, this.album1,       Valoracion.LIKE)
		.valorar(usuario1, this.canciones[3], Valoracion.DISLIKE)
		.valorar(usuario2, this.canciones[1], Valoracion.DISLIKE);
	fonoteca.mostrarValoraciones(usuario1);
	fonoteca.mostrarValoraciones(usuario2);
  }
}
```

## Salida esperada:

```
music.exceptions.ExcepcionCancionRepetida: La cancion Resistire esta repetida
music.exceptions.ExcepcionCancionRepetida: La cancion Tear it up esta repetida
Valoraciones de smelero
CANCION: Radio ga ga [DISLIKE]
ALBUM: The Works, ARTISTA: Queen, DURACION: 13:24, ESTILO: ROCK [LIKE]
CANCION: Tear it up [LIKE]
CANCION: It's a hard life [LIKE]
CANCION: Resistire [DISLIKE]
Valoraciones de mcuevas
CANCION: Tear it up [DISLIKE]
```

## Apartado 4. Recomendación de música popular (2 puntos)

En este apartado y en el siguiente añadiremos recomendadores de música a la fonoteca, sobre la base del componente de valoración creado en el apartado 3. De nuevo, diseñaremos estos sistemas de recomendación de manera general – mediante el uso de interfaces – para permitir su uso no sólo para recomendar música, si no también cualquier otro tipo de elemento (por ejemplo, películas, libros, destinos de viaje).

Una vez que los usuarios han valorado canciones y álbumes, la fonoteca debe recomendar a los usuarios nuevas canciones y álbumes que podrían interesarles. Las recomendaciones las generará una clase adicional, a partir de las valoraciones previas de los usuarios. Para permitir implementar distintas estrategias de recomendación, y para poder recomendar distintos tipos de objetos, la clase que generará las recomendaciones debe implementar la siguiente interfaz:

```
public interface IRecomendador extends IAlmacenValoraciones {
	Collection<Recomendacion> getRecomendaciones(IUsuario usuario);
	void setCorte(double corte);
}
```

Los métodos de esta interfaz deben tener el siguiente comportamiento:

- `getRecomendaciones(IUsuario)`: devuelve una lista de recomendaciones para un usuario. Como se explica a continuación, cada recomendación incluye el elemento recomendado, y la confianza en la recomendación (un valor entre 0 y 1). Las recomendaciones devueltas deben estar ordenadas por su nivel de confianza, de mayor a menor confianza. Sólo se devolverán las recomendaciones que tengan un mínimo nivel de confianza, establecido mediante el método `setCorte`. Además, no deben recomendarse elementos que el usuario haya valorado con anterioridad.
- `setCorte(double)`: establece la confianza mínima que una recomendación debe tener para ser recomendada.

La interfaz `IRecomendador` usa la clase `Recomendacion` que incluye un elemento recomendado y la confianza de la recomendación. La clase – que debes completar – tiene que implementar la interfaz `Comparable` de la librería estándar de Java, para permitir hacer comparaciones.

```
public class Recomendacion implements Comparable<Recomendacion> {
	private IRecomendable elemento;
	private double confianza;
	// Completa esta clase
}
```

En este apartado, se pide crear la clase `RecomendadorPopularidad`, que implementará la interfaz `IRecomendador`. Esta clase debe recomendar a los usuarios los elementos (canciones y álbumes) más populares en la comunidad de usuarios. Por tanto, las clases `Cancion` y `Album` deberán implementar la interfaz `IRecomendable`. La popularidad de un elemento recomendable se calculará sumando 1 por cada `LIKE` que haya recibido el elemento, y restando 0.5 por cada `DISLIKE` recibido. Para realizar los cálculos, recuerda que la valoración de una canción puede ser explícita, o recibirla del álbum que la contiene. La confianza para recomendar un elemento se calculará como el cociente de su popularidad entre el número total de usuarios. Por ejemplo, si una canción tiene popularidad 3, y hay 5 usuarios registrados, el nivel de confianza al recomendar la canción será 3/5=0.6. Como recoge la definición previa del método `getRecomendaciones(IUsuario)`, sólo se recomendarán elementos por encima del nivel de confianza fijado, y un usuario no puede recibir recomendaciones de elementos que ha valorado. A continuación, tienes un programa de prueba con la salida esperada.

```
public class FonotecaTesterPopularidad extends FonotecaTesterErrores {
  private Album album2;
  public static void main(String[] args) {
	FonotecaTesterPopularidad main = new FonotecaTesterPopularidad();
	Fonoteca fonoteca = new Fonoteca(0.4);  // corte 0.4 para el recomendador
	main.crearMusica(fonoteca);
	main.recomendaciones(fonoteca);
  }
  public void crearMusica(Fonoteca fonoteca) {
	super.crearMusica(fonoteca);
	try {
		this.album2 = fonoteca.crearAlbum("Resistire", "Duo dinamico", canciones[3], canciones[4]);
	} catch (ExcepcionCancionRepetida e) {
		e.printStackTrace();
	}
  }
  protected void recomendaciones(Fonoteca fonoteca) {
	Usuario[] usuarios = { fonoteca.registrarUsuario("Sonia Melero Vegas",   "smelero"),
			       fonoteca.registrarUsuario("Miguel Cuevas Alonso", "mcuevas"),
			       fonoteca.registrarUsuario("Lucas Varas Peinado",  "lvaras")};
	fonoteca.valorar(usuarios[0], album1,       Valoracion.LIKE)
		.valorar(usuarios[0], canciones[3], Valoracion.LIKE)
		.valorar(usuarios[1], canciones[0], Valoracion.LIKE)
		.valorar(usuarios[1], canciones[1], Valoracion.LIKE)
		.valorar(usuarios[1], album2,       Valoracion.LIKE)
		.valorar(usuarios[2], canciones[2], Valoracion.LIKE)
		.valorar(usuarios[2], canciones[3], Valoracion.LIKE)
		.valorar(usuarios[2], canciones[1], Valoracion.DISLIKE);
	for (Usuario u: usuarios) {
		System.out.println("---------------------------");
		fonoteca.mostrarRecomendaciones(u);
	}
  }
}
```

**Salida esperada:**

```
music.exceptions.ExcepcionCancionRepetida: La cancion Resistire esta repetida
music.exceptions.ExcepcionCancionRepetida: La cancion Tear it up esta repetida
--------------------------
RECOMENDACIONES PARA: smelero
--------------------------
RECOMENDACIONES PARA: mcuevas
CANCION: It's a hard life [0,67]
--------------------------
RECOMENDACIONES PARA: lvaras
CANCION: Radio ga ga [0,67]
```

## Apartado 5. Recomendación de música favorita de usuarios afines (2 puntos)

Para ilustrar la utilidad de las interfaces, en este apartado crearemos un segundo recomendador (`RecomendadorAfinidad`) que implemente la interfaz `IRecomendador`. Este recomendador sugerirá a cada usuario canciones y álbumes que les han gustado a otros usuarios con gustos similares. Además, configuraremos la fonoteca en el constructor con el recomendador que queremos.

Para generar las recomendaciones para un usuario, el recomendador comenzará calculado la afinidad de ese usuario con cada uno de los demás usuarios. La afinidad entre dos usuarios se calculará sumando 1 por cada elemento recomendable que los dos usuarios hayan valorado con `LIKE`, sumando 0.5 por cada elemento que ambos hayan valorado con `DISLIKE`, y restando 0.5 por cada elemento que uno haya valorado con `LIKE` y el otro con `DISLIKE`. A continuación, el recomendador asignará a los elementos que le gustan a otro usuario un nivel de confianza igual al cociente entre la afinidad con ese otro usuario, y el número total de elementos recomendables en la fonoteca. Por ejemplo, si la afinidad con otro usuario es 3, y hay 8 elementos recomendables en la fonoteca, el nivel de confianza para recomendar los elementos que le gustan al otro usuario sería 3/8=0.375. Si varios usuarios han valorado favorablemente el mismo elemento, se tomará el que tenga mayor nivel de confianza.

A continuación, tienes un programa de prueba con la salida esperada. Debes conservar los constructores de `Fonoteca` que ya tienes, que utilizarán por defecto el recomendador creado en el apartado anterior. El nuevo constructor parametriza la fonoteca con un recomendador conforme a `IRecomendador`, y permite pasarle nuevos recomendadores que se construyan en el futuro.

```
public class FonotecaTesterAfinidad extends FonotecaTesterPopularidad {
  public static void main(String[] args) {
    FonotecaTesterAfinidad main = new FonotecaTesterAfinidad();
    Fonoteca fonoteca = new Fonoteca (new RecomendadorAfinidad(0.2));  // Con rec. afinidad de corte 0.2
    main.crearMusica(fonoteca);
    main.recomendaciones(fonoteca);
  }
}
```

**Salida esperada:**

```
music.exceptions.ExcepcionCancionRepetida: La cancion Resistire esta repetida
music.exceptions.ExcepcionCancionRepetida: La cancion Tear it up esta repetida
--------------------------
RECOMENDACIONES PARA: smelero
ALBUM: Resistire, ARTISTA: Duo dinamico, DURACION: 7:10, ESTILO: SIN ESTILO [0,43]
CANCION: Dos corazones [0,43]
--------------------------
RECOMENDACIONES PARA: mcuevas
ALBUM: The Works, ARTISTA: Queen, DURACION: 13:24, ESTILO: ROCK [0,43]
CANCION: It's a hard life [0,43]
--------------------------
RECOMENDACIONES PARA: lvaras
ALBUM: The Works, ARTISTA: Queen, DURACION: 13:24, ESTILO: ROCK [0,21]
CANCION: Radio ga ga [0,21]
```

### Nota:

- En este apartado, debes rediseñar las clases de los recomendadores, sacando factor común de los elementos comunes en ambos recomendadores, y evitando la repetición de código.

---

## Comentarios adicionales:

1. No olvides que, aunque evaluaremos la corrección funcional de la práctica, un aspecto fundamental en la evaluación será **la calidad del diseño** (claro, fácil de entender, extensible, flexible, y evitando redundancias).
2. Recuerda organizar el código en **paquetes**.
3. **Crea programas de prueba** para ejercitar el código que escribas en cada apartado. Deberás entregar estos programas junto con tu código. Puedes usar JUnit para crear las pruebas.

## Normas de Entrega:

- La entrega la realizará uno de los estudiantes de la pareja, a través de Moodle.
- Debes entregar el **código Java** de los apartados, adecuadamente **comentado**, y la **documentación** generada con *javadoc*. Entrega también todos los **programas de prueba** que hayas creado.
- Entrega un **diagrama de diseño** de toda la práctica (en PDF) junto con una breve explicación.
- Se debe entregar un único fichero ZIP / RAR con todo lo solicitado, que deberá llamarse de la siguiente manera: `GR<numero_grupo>_<nombre_estudiantes>.zip`. Por ejemplo, Roberto y Carlos, del grupo 2261, entregarán el fichero: `GR2261_RobertoCarlos.zip`.