# Programación Orientada a Objetos

## Trabajo Práctico Especial

Facultad de Ciencias Exactas - UNCPBA
Ingeniería de Sistemas

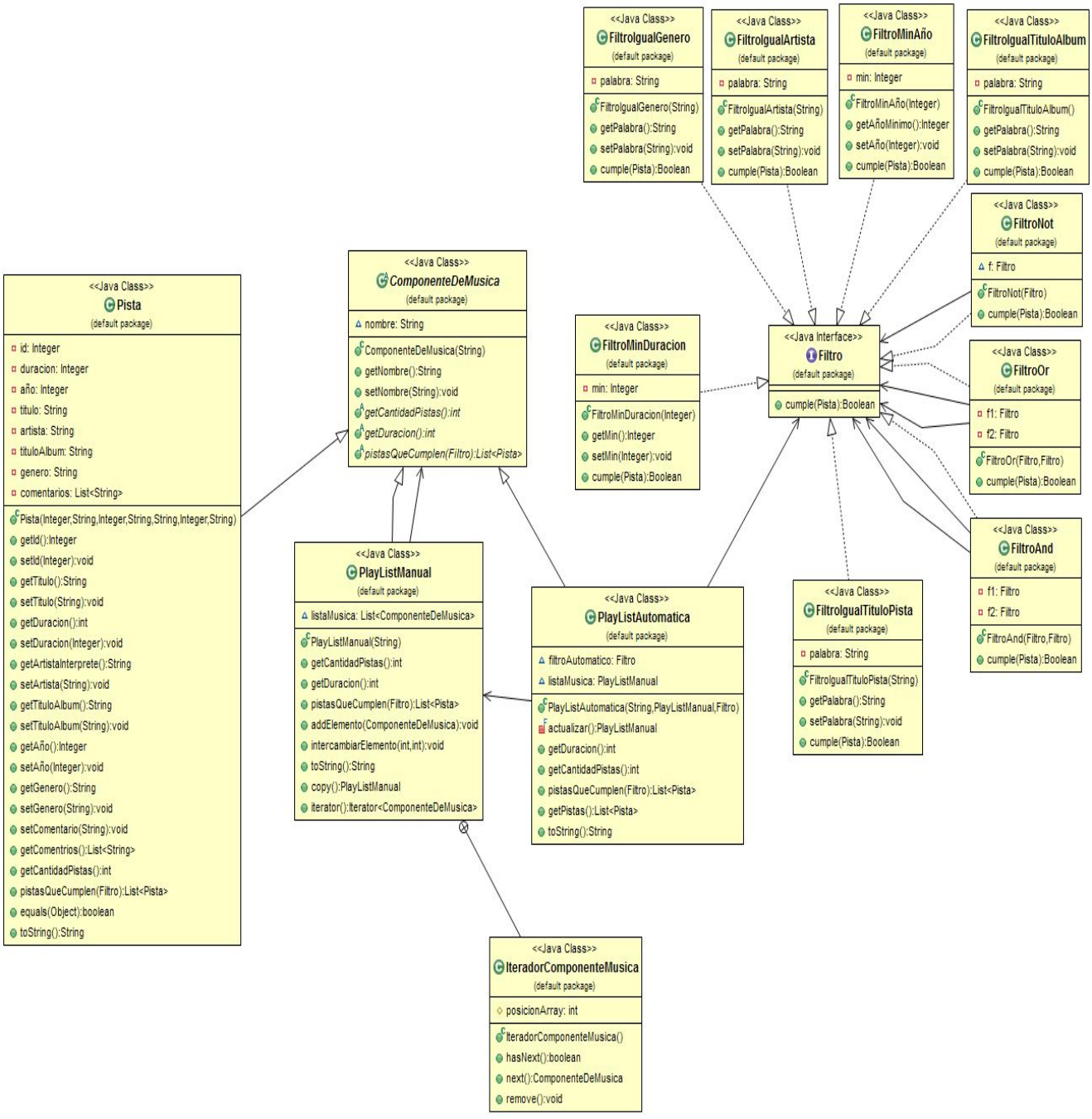

FECHA DE ENTREGA: 9/05/2016
GRUPO NÚMERO: 8
INTEGRANTES:

Hugo Avendaño - 247860

Ayudante:

Dr. Mauricio Arroqui

# Diagrama de Clases



**Análisis del Diagrama de Clases**

Las clases e interfaces que reconoci para implementar una solución orientada a objetos en el trabajo práctico fueron las siguientes:

***ComponenteDeMusica.*** Es una clase abstracta que tiene como único atributo un nombre del tipo String. Los métodos que implementa esta clase son el *setNombre*, que sirve para definir el título de una pista o el nombre de una playlist tanto las manual como automática, y el *getNombre* definido para obtener el valor del atributo nombre. Por su parte, los métodos abstractos *getCantidadPistas*, *getDuracion*, *pistasQueCumplen* son implementados por las sublcases **Pista**, **PlayListManual**, **PlayListAutomatica**.

***Pista.*** Es una subclase de **ComponenteDeMusica,** que almacena en variables de clase los atributos :

*id*: Integer.
*duracion* (en segundos): Integer
*artista*: String.
*tituloAlbum*: String
*año:* Integer
*género* (rock, pop, melódico, etc.): String
*comentarios*: List<String>

La clase Pista tiene los métodos getters and setters lo que proporciona la funcionalidad de cambiar cualquiera de los atributos mencionados.

Además implementa los métodos abstractos de la superclase de la cual hereda. Estos son:

- *public int getDuracion():* retorna la cantidad de segundos que dura una pista
- public int getCantidadPistas(): retorna 1, ya que la cantidad de pistas de una pista siempre es 1.
- *public List<Pista> pistasQueCumplen(Filtro F*): este metodo recibe como parametro un filtro(requisito), en caso que la pista cumpla

con el requisito esta se retorna dentro de una lista y si no cumple retorna una lista vacía.

Por otro lado se sobreescriben los métodos, que se heredan ***java.lang.Object***, *:*

- *public String toString():* este método tiene como objetivo mostrar la información que completa el objetos pista;
- *public public boolean equals(Object o):* este método retorna si dos pistas son iguales, es decir si el contenido de dos pistas son iguales.

**PlayListaManual.** Es una subclase de Componente de musica, la cual tiene solo como atributo una lista de **ComponenteDeMusica** *listaMusica*. De esta forma se cumple con el requisito del enunciado de que "una playlist incluya como uno de sus elementos otra playlist", utilizando el patron de diseño composite que nos ofrece la posibilidad de construir objetos complejos a partir de otros más simples y similares entre sí, gracias a la composición recursiva. Esta clase implementa los métodos abstractos de la superclase de la cual hereda. Estos son:

- *public int getCantidadPistas():* este metodo devuelve la cantidad de pistas que contiene una playlist. Para ello recorre la lista de **Componentes de musica** y por cada componente calcula la cantidad de pista que tiene.
- public int getDuracion(): este metodo retorna la duración de un playlist en segundos. Al igual que el método anterior, para realizar el cálculo recorre la lista y por cada componente de la misma calcula la duración.
- *public List<Pista> pistasQueCumplen(Filtro f):* este metodo retorna una lista(*cumplen*) con el conjunto de pistas de la **PlayListAutomatica** que satisfacen un determinado criterio(filtro). Para ello se recorre la *listaMusica* y verifica por cada componente si cumple con el **Filtro** pasado por parámetro y en caso de que cumpla y no se encuentre añadido a la lista

solución(*cumplen)* es añadido, es decir que la lista que se retorna no tiene pistas repetidas.

**PlayListManual** además implementa la interface iterable que está incluida en el api de Java, en concreto en el paquete **java.lang.** Los método que nos obliga a implementar la interfaz Iterable son:

- *public Iterator<**ComponenteDeMusica**> iterator()*

Para poder devolver un objeto de tipo Iterator (que es algo a lo que al fin y al cabo nos obliga la interface Iterable) necesitamos instanciar un objeto Iterator definiendo una clase interna dentro de la clase PlayListManual denominada **IteradorComponenteMusica**, que implementará la interface Iterator, y que nos permitirá devolver una instancia de Iterator para nuestra clase **PlayListManual**.El acceso elegido para crear la clase es **IteradorComponenteMusica** protected. Porque esta clase no tiene interés que sea visible desde otras clases. Únicamente interesa que sea visible desde la clase Persona o subclases de la clase Persona.La interface Iterator (del paquete java.util) a su vez nos obliga a implementar al menos 3 métodos que son:

- *public boolean hasNext()*:este metodo retorna un valor boolean indicando si el iterador tiene un siguiente elemento
- *public ComponenteDeMusica next()*: devuelve el siguiente elemento del iterador
- *public void remove(): elimina el elemento que se obtiene del iterador*

Por último, **PlayListManual** tiene implementados los siguientes métodos.

- *public void addElemento(ComponenteDeMusica c):* permite agregar un **ComponenteDeMusica** a la playlist.
- *public void intercambiarElemento(int elemento1, int elemento2):* este método recibe por parámetro dos enteros que referencian la posición de 2 elementos para realizar un intercambio de posiciones en la estructura.
- *public PlayListManual copy() :* este método permite obtener una copia de una **PlayListManual.** La nueva playlist se crea con el

nombre “Copia de NOMBRE_PLAYLIST”, donde NOMBRE_PLAYLIST es el nombre de la lista original. Al duplicar una playlist, no se duplican los elementos que contiene (sean pistas u otras playlists).

- *public String toString() :*este método tiene como objetivo mostrar la información que completa el objeto PlayListManual.

**PlayListAutomatica.** Esta subclase define un caso especial de playlist cuyo contenido se actualiza dinámicamente (en cada acceso a la misma) con referencia a las pistas que cumplen con determinado criterio de búsqueda (simple o compuesto). Las operaciones de la playlist automática deben tener en cuenta el estado actual de una **PlayListManual**, es por ello que la clase guarda como atributos privados una **PlayListManual** de referencia *listaMusica* y un **Filtro** *filtroAutomatico* y posee el método privado:

- *private final PlayListManual actualizar():* este método devuelve una playlist a partir de la PlayListManual *listaMusica y el Filtro filtro automático.* Es utilizado por los metodos *getCantidadPistas(), getDuracion(), public List<Pista> pistasQueCumplen(Filtro f).*

Esta clase implementa los métodos abstractos de la superclase de la cual hereda. Estos son:

- *public int getCantidadPistas():* este método devuelve la cantidad de pistas que contiene.
- public int getDuracion(): este metodo retorna la duración de un playlist en segundos.
- *public List<Pista> pistasQueCumplen(Filtro f):* este metodo retorna una lista(*cumplen*) con el conjunto de pistas de la **PlayListAutomatica** que satisfacen un determinado criterio(filtro).

Ademas esta subclase tiene implementados los metodos los siguientes metodos:

- *public List<Pista> getPistas()*: retorna la lista de pistas que contiene la playlist, es decir las pistas que cumplen con el filtroAutomatico.
- *public String toString()*: este método tiene como objetivo mostrar la información que completa el objeto PlayListAutomatica.

**Filtro.** Es una interface que contiene un solo método abstracto(*public abstract Boolean cumple(Pista p)*) en que se especifica si una pista cumple con un criterio pero no su implementación. Para ello se crearon las distintas clases(**filtroIgualArtista, filtroIgualGenero, filtroIgualTituloAlbum, filtroIgualTituloPista, filtroMinAño, FiltroMinDuracion**) que implementan la interface **Filtro**.

**FiltroNot.** Esta clase también implementa la interface **Filtro** pero a diferencia de las mencionadas anteriormente, tiene un atributo privado del tipo Filtro y cuando se implementa *public Boolean cumple(Pista p) retorna la negación del filtro que se tiene como atributo.*

**FiltroAnd.** Esta clase también implementa la interface **Filtro**. Tiene dos atributo privado del tipo Filtro y cuando se implementa *public Boolean cumple(Pista p) retorna el "and" de los filtros que se tienen como atributos.*

**FiltroOr.**Esta clase también implementa la interface **Filtro**. Tiene dos atributo privado del tipo Filtro y cuando se implementa *public Boolean cumple(Pista p) retorna el "or" de los filtros que se tienen como atributos.*