Mediateca

*Patterns Java Project*

Mediateca es una aplicación en Java que permite la gestión de préstamos de libros, comics, discos y películas a sus usuarios con una interfaz simple y amigable.

Adrián de Juan Lora

Ignacio Ribera Díaz

Mediateca

Patterns Java Project

Indice.

Contenido

[Enunciado y requisitos 2](#_Toc504087137)

[Instrucciones de ejecución y manual de usuario 3](#_Toc504087138)

[Instrucciones de ejecución. 3](#_Toc504087139)

[Manual de usuario 4](#_Toc504087140)

[Diseño de la aplicación (uml) y patrones utilizados 5](#_Toc504087141)

[Diseño de la aplicación (UML) 5](#_Toc504087142)

[Diagrama de clases 5](#_Toc504087143)

[Diagrama de casos de uso 5](#_Toc504087144)

[Patrones utilizados 6](#_Toc504087145)

[Fachada 6](#_Toc504087146)

[Estrategy 6](#_Toc504087147)

[Singleton 6](#_Toc504087148)

[Proxy 6](#_Toc504087149)

[Iterator 6](#_Toc504087150)

[Prototype 6](#_Toc504087151)

[Observer 6](#_Toc504087152)

[Factory method 7](#_Toc504087153)

# Enunciado y requisitos

Se desea crear un sistema gestor de una mediateca. La mediateca contiene

distintos artículos como libros, comics, cinematografía, y audio. El

sistema debe diferenciar los diferentes artículos según sus diferentes

características.

Los usuarios deben de ser capaces de alquilar artículos durante un tiempo

determinado. El administrador debe de ser capaz de gestionar tanto el alta y

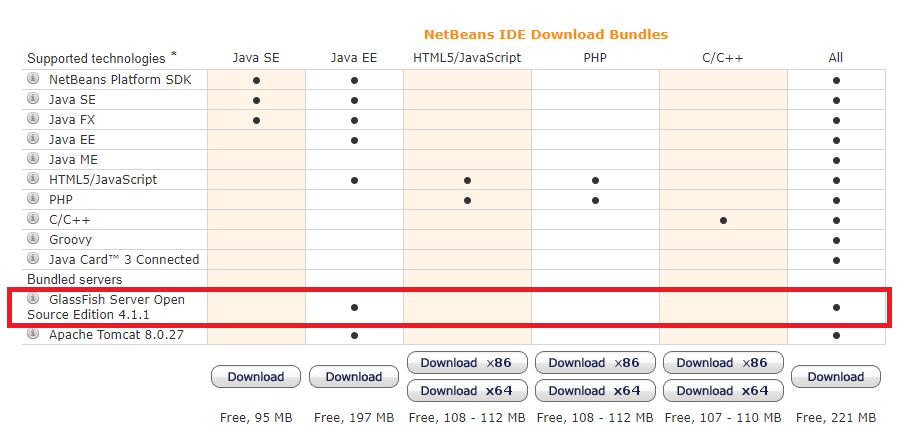
baja, hacer administrador a un usuario o darle de baja.

Los artículos y usuarios están registrados en una base de datos.

# Instrucciones de ejecución y manual de usuario

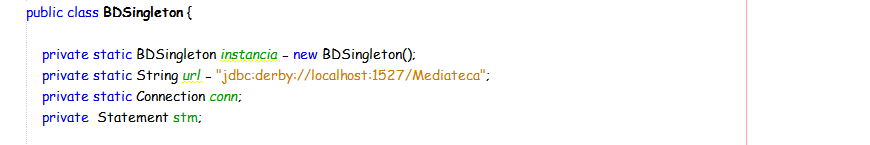
## Instrucciones de ejecución.

Para la instalación del programa, descargar el instalador de NetBeans con todas las extensiones o la versión EE e instalarlo. También tiene la opción de instalar simplemente el servidor GlassFish en su distribución actual de NetBeans.



Una vez instalado NetBeans y el servidor, se deberá crear una base de datos de java y recrear las tablas con los archivos .grasp proporcionados en la práctica. Con la base de datos ya creada, utilizar el script proporcionado para la carga inicial de datos para la prueba.

Una vez hecho esto, importar el proyecto y cambiar la ruta de la base de datos (ir a la clase BDSingleton y cambiar la dirección en el String url.



Una vez hecho esto, simplemente ejecutamos el Home.java de la carpeta interfaz y podremos empezar a utilizar la aplicación.

## Manual de usuario

### Gestión usuarios

#### Creación de usuario

#### Borrar usuario

#### Hacer administrador

### Gestión de artículos

#### Insertar articulo

#### Dar de baja artículo

### Préstamo de artículo

#### Alquilar artículo

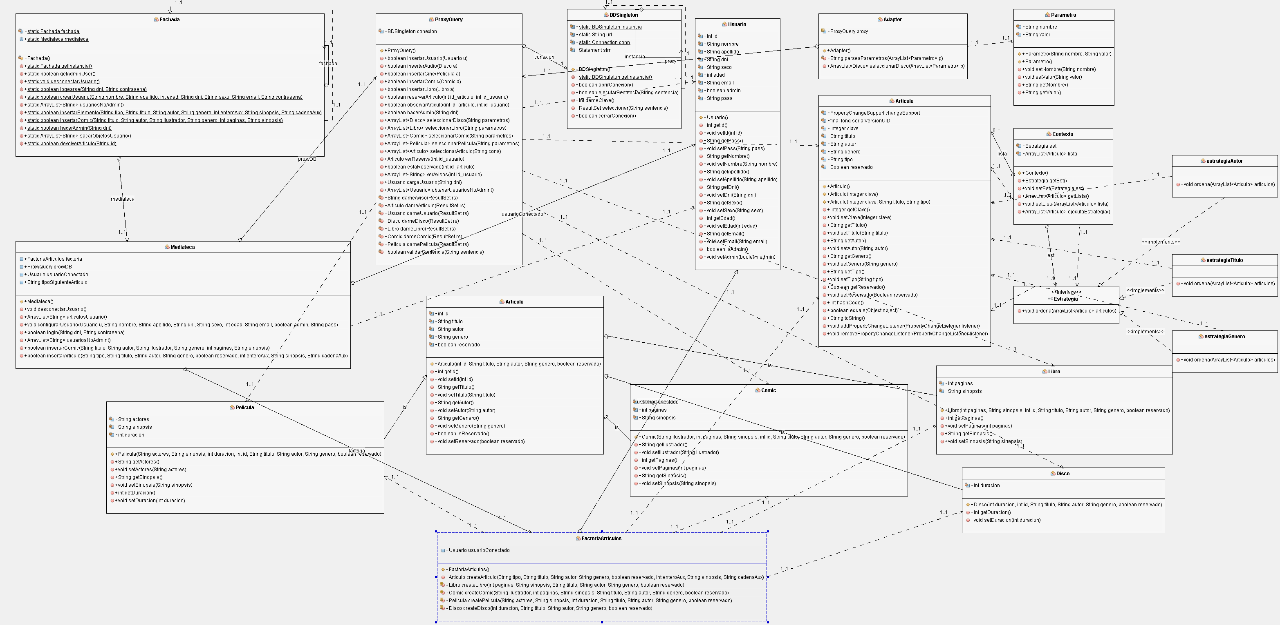
#### Devolver artículo

# Diseño de la aplicación (uml) y patrones utilizados

## Diseño de la aplicación (UML)

### Diagrama de clases

A continuación, aparece el diagrama de clases del programa.

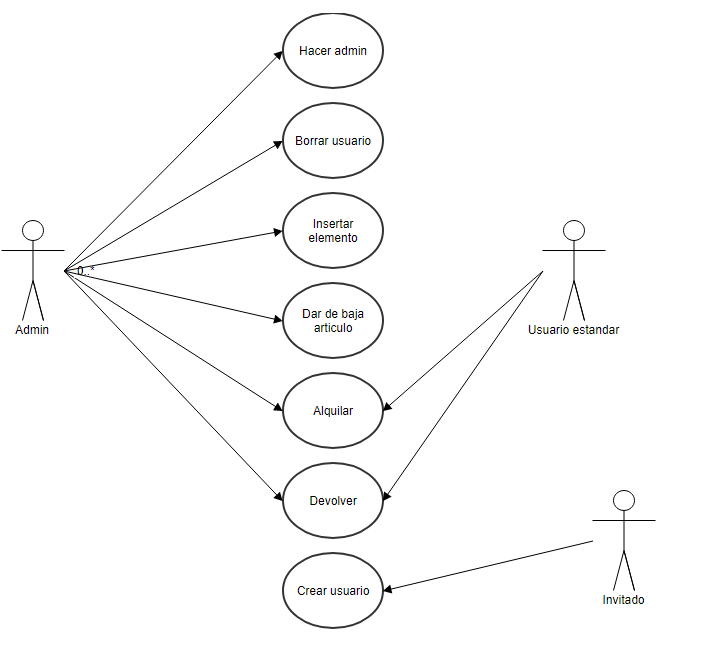


El programa está estructurado de la siguiente manera:

* Hay una clase Articulo no abstracta (utilizamos instancias para la base de datos) de la que heredan los 4 tipos de articulos: libros, comics, discos y películas. De esta manera, los cuatro compartirán los atributos id, titulo, autor y género.
* Estas clases serán instanciadas por un método en la clase fachada, el cual devuelve un objeto de tipo Articulo.
* La Fachada es una clase estática en la que está almacenada la mediateca, y es la clase que da interfaz unificada del programa.
* La mediateca contiene el objeto proxy, el cual nos comunicará con el objeto DBSingleton que accede a la base de datos. También tiene un al usuario logeado en ese momento y un objeto factoría para construir los artículos.
* La clase usuario se inicializa con unos valores predeterminados, que luego se configuran para construir cada usuario concreto y cumplir el patrón prototype.
* La clase ProxyQueryDB es la clase que tiene acceso a la clase estática DBSingleton, para unificar el acceso a la base de datos y solo instanciar una conexión a la base de datos (DBSingleton):
* La clase Parametro se usa para las búsquedas en base de datos, para meter condiciones de búsqueda. Tiene nombre y valor, por ejemplo: Nombre:titulo Valor:DragonBall.
* La clase Adapter se preocupa de saber en qué tabla hay que buscar y cuál es la sentencia que hay que enviar para la búsqueda.
* La clase Estrategia implementa 3 estrategias diferentes para ordenar las búsquedas.

### Diagrama de casos de uso

Ahora presentaremos los casos de uso planteado en nuestro programa, aquí tenéis un diagrama explicativo.



Crear usuario: un usuario no registrado, podrá registrarse en la mediateca y obtendrá una cuenta de nivel no administrador.

Hacer administrador: un usuario administrador podrá acceder a la lista de usuarios no administrador y darle estos derechos.

Borrar usuario: un usuario administrador podrá borrar un usuario (no administrador) de la aplicación.

Insertar elemento: un usuario administrador podrá insertar nuevos artículos a la mediateca (de los 4 tipos).

Dar de baja articulo: un usuario administrador podrá quitar un artículo del sistema.

Alquilar: un usuario (tanto administrador como normal) podrá alquilar un articulo a través de la aplicación.

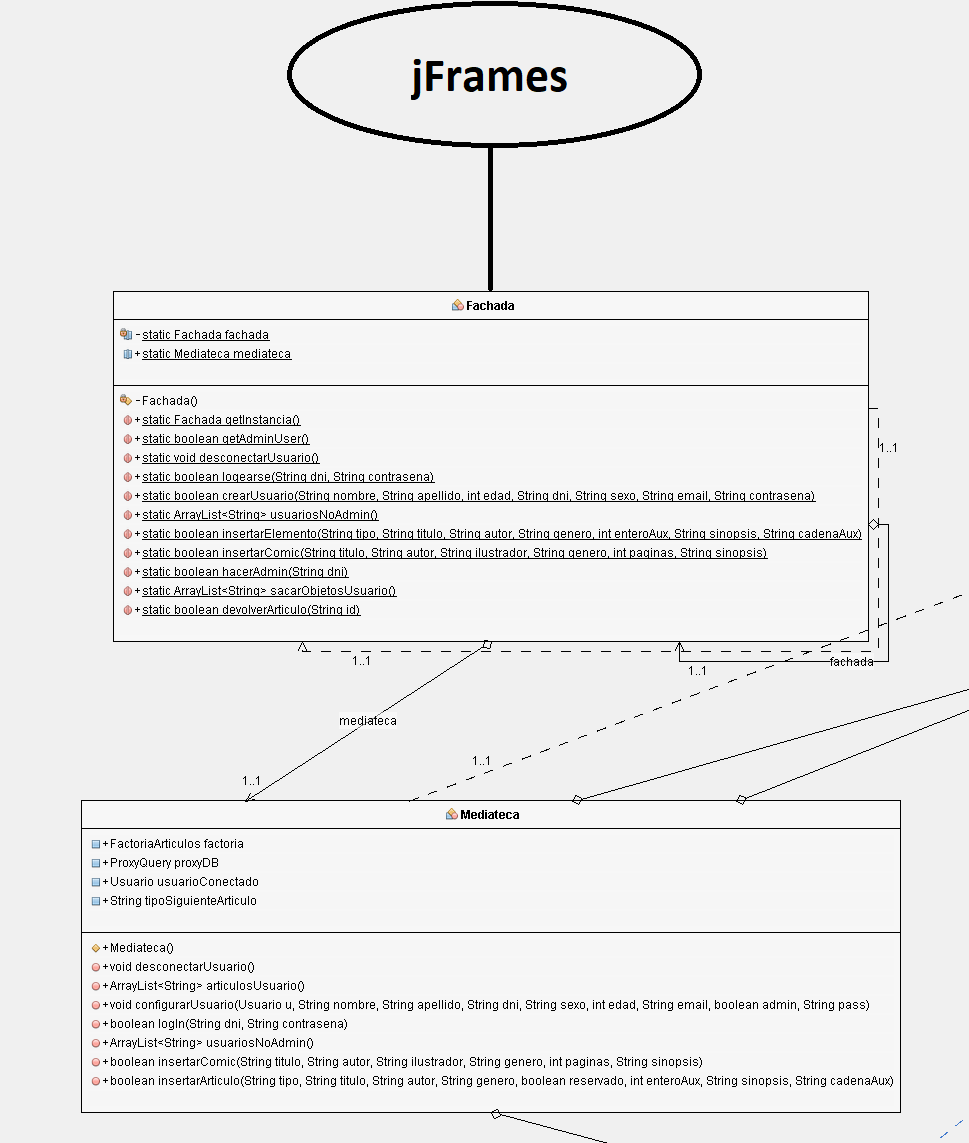
Devolver: un usuario (tanto administrador como normal) podrá devolver el artículo que tiene alquilado.

## Patrones utilizados

En esta parte explicaremos todos los patrones utilizados en la práctica.

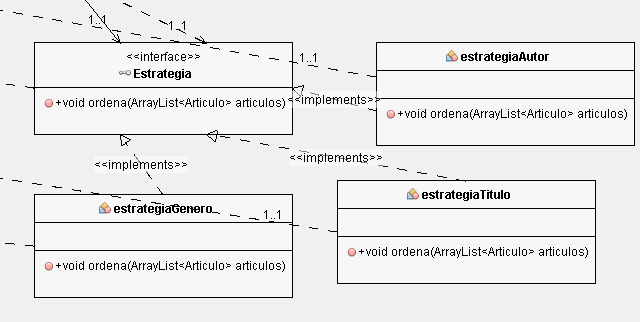
### Fachada

Para la implementación de este patrón hemos definido una clase estática fachada, en la que se encontraba instanciado un objeto mediateca. De esta manera, conectamos la mediateca y las ventanas de la interfaz gráfica unificando la interfaz en un solo objeto, llamando a funciones de la fachada exclusivamente. Esto ha sido combinado con el patrón singleton, ya que solo se puede instanciar un objeto de tipo Fachada (clase de tipo static).



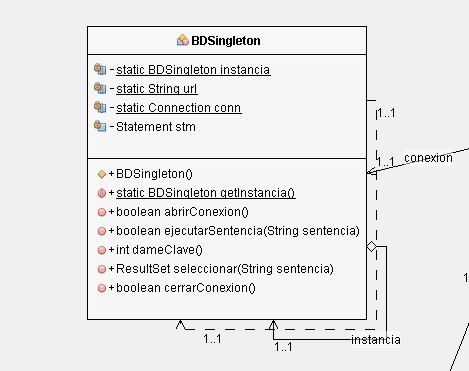
### Estrategy

La clase Estrategia tiene un método ordenar y puede implementar 3 estrategias distintas: estrategiaGenero, estrategiaTitulo y estrategiaAutor.



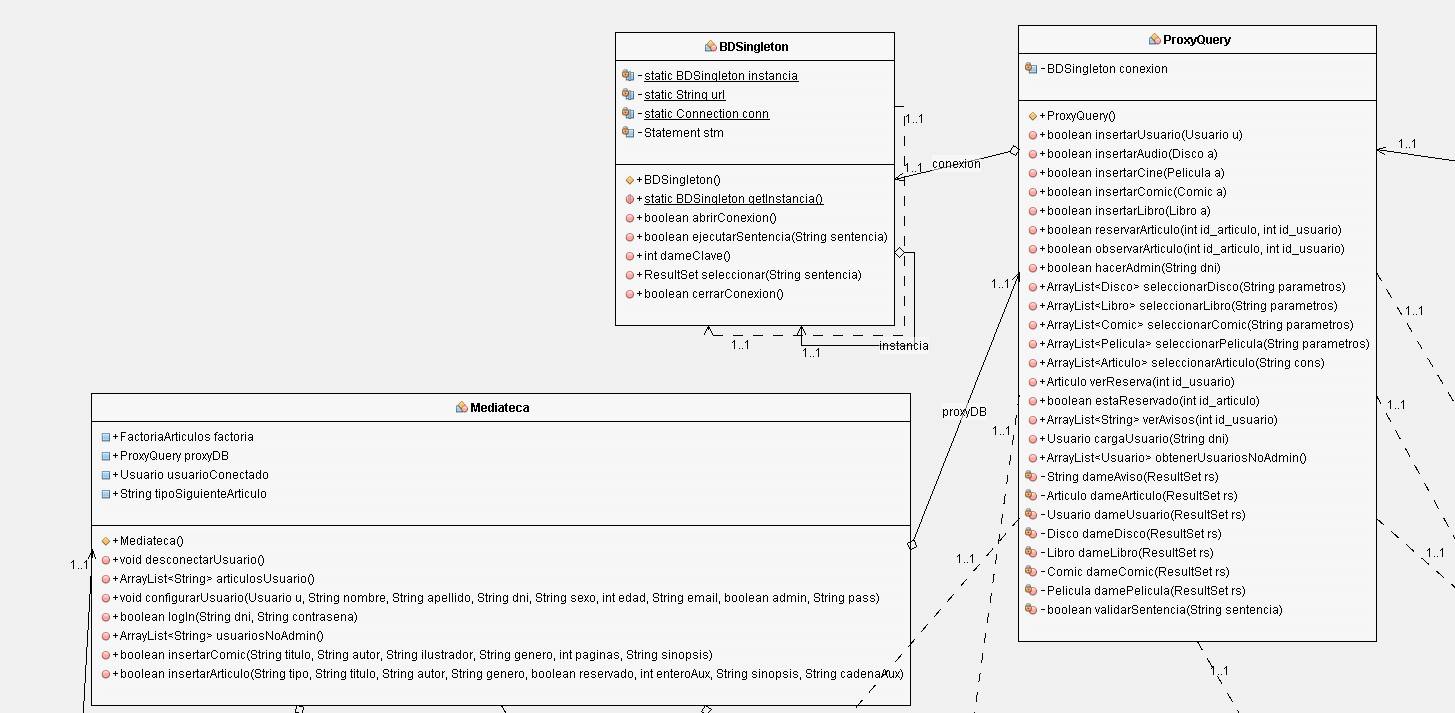
### Singleton

Hemos utilizado este patrón en dos ocasiones. La primera que pensamos fue para el acceso a la base de datos, para limitarlo a un único objeto (DBSingleton de tipo static). La segunda fue para comunicar el programa con la interfaz, implementamos el singleton a la fachada, para que todos los jFrame tuvieran acceso a una única (y la misma) mediateca (instanciada dentro de la fachada).



### Proxy

Para limitar el acceso a base de datos, se ha creado una clase ProxyDB, que es la única clase con acceso directo a la clase BDSingleton, que es la que se comunica con la base de datos. Este patrón ha sido combinado con el Singleton, de tal manera, que DBSingleton solo pueda instanciarse una vez, y así tener un único acceso a la base de datos.

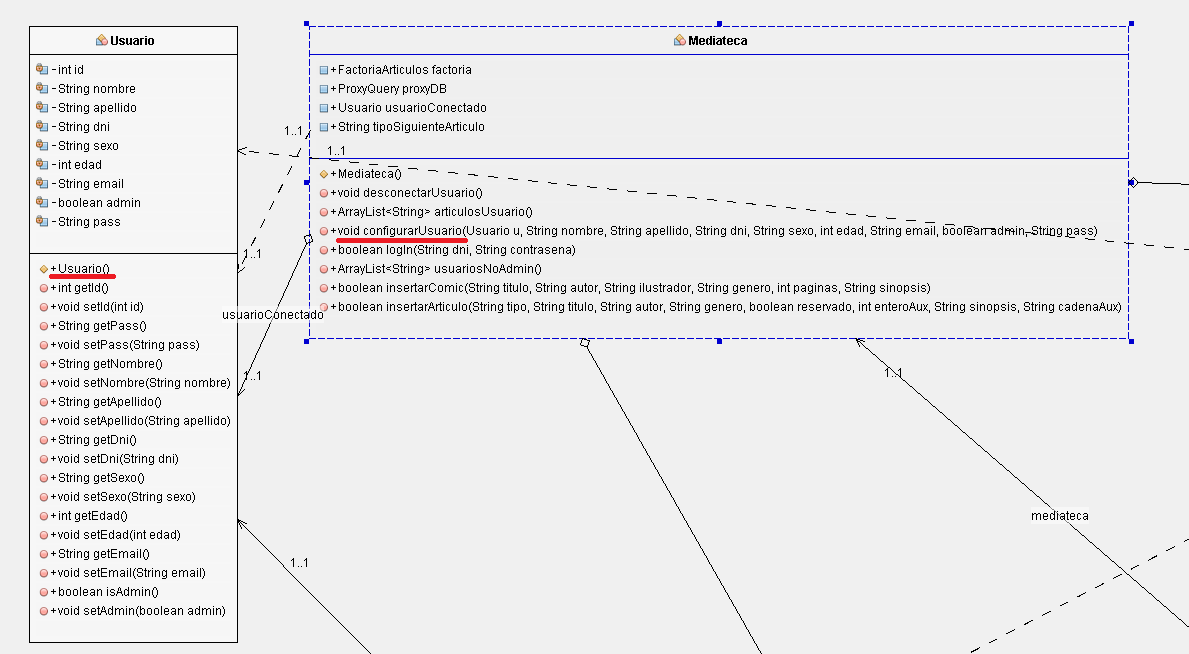


### Iterator

Este patrón es el más sencillo de implementar, simplemente, cuando tenemos que recorrer una colección (en este caso hemos usado ArrayList), declaramos un iterador (Iterator iterador = arrayUsuarios.iterator();) y tenemos acceso de manera cómoda a los elementos del ArrayList.

### Prototype

El patrón prototype ha sido implementado para la creación de usuarios, la clase usuario tiene un constructor sin parámetros que instancia siempre un usuario modelo. También dispone de un método configurarUsuario, al que se le pasan los parámetros del nuevo usuario.

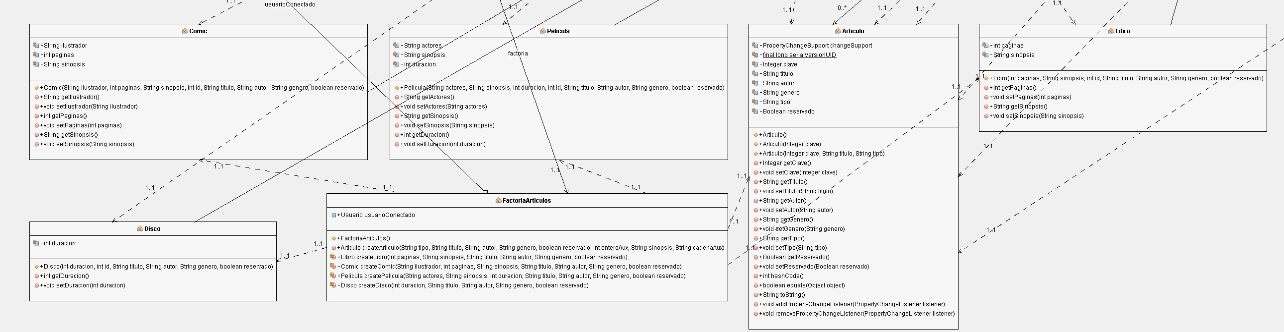


### 

### Observer

### Factory method

Tenemos 4 artículos diferentes, pero todos heredan de Articulo. Por esto, hemos creado una clase Factoría, que tiene 1 función para instanciar cada uno de los elementos, y luego una función genérica que combina las 4. De esta manera, con una sola función, instanciamos artículos de los 4 tipos, sin tener que usar 4 funciones diferentes.



### Adapter

