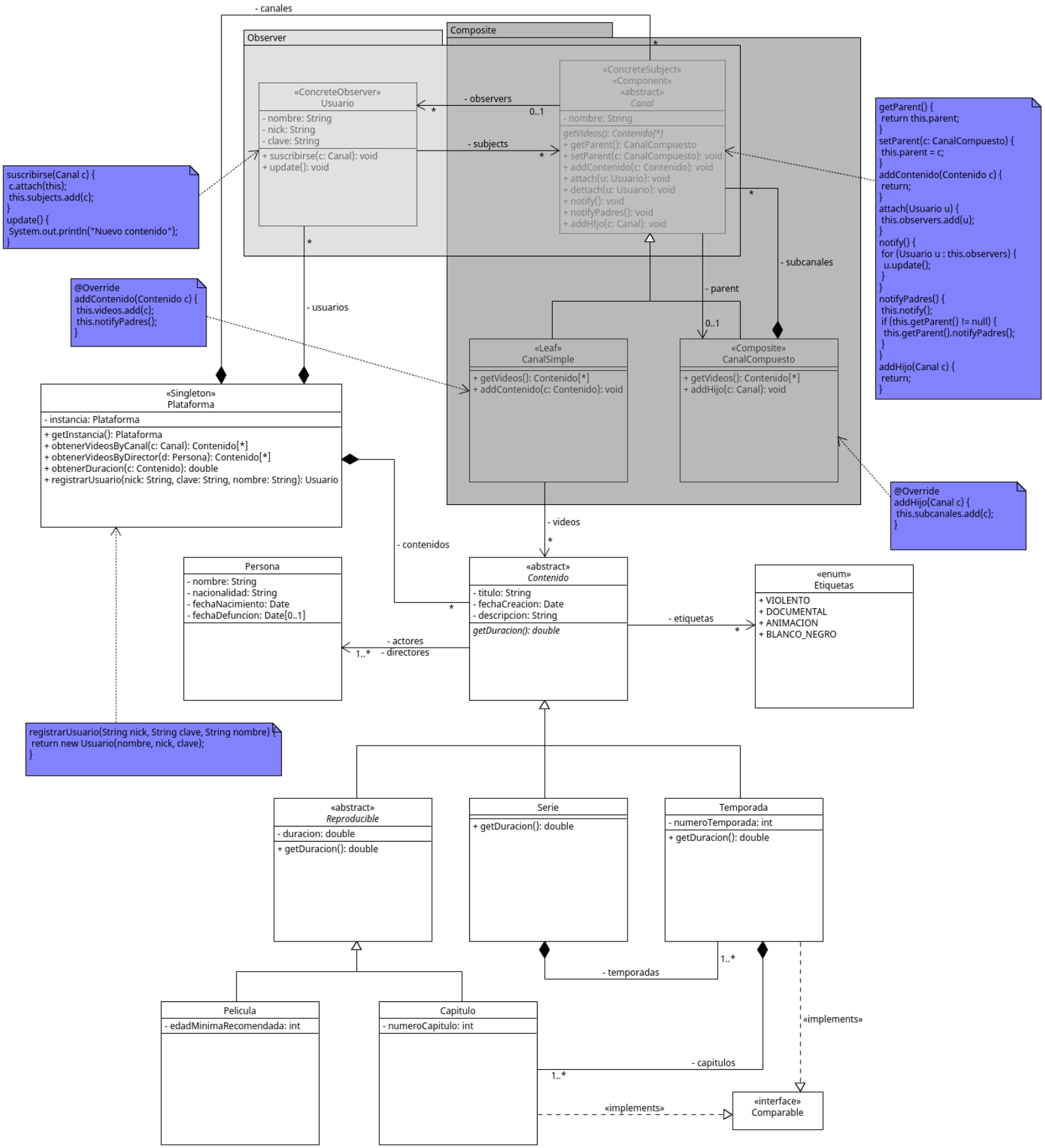
- canales
Observer
Composite
«ConcreteObserver»
Usuario
- nombre: String
- nick: String
- clave: String
+ suscribirse(c: Canal): void
+ update(): void
- observers
0..1
- subjects
«ConcreteSubject»
«Component»
«abstract»
Canal
- nombre: String
getVideos(): Contenido[*]
+ getParent(): CanalCompuesto
+ setParent(c: CanalCompuesto): void
+ addContenido(c: Contenido): void
+ attach(u: Usuario): void
+ dettach(u: Usuario): void
+ notify(): void
+ notifyPadres(): void
+ addHijo(c: Canal): void
- subcanales
- parent
0..1
- usuarios
«Leaf»
CanalSimple
+ getVideos(): Contenido[*]
+ addContenido(c: Contenido): void
«Composite»
CanalCompuesto
+ getVideos(): Contenido[*]
+ addHijo(c: Canal): void
«Singleton»
Plataforma
- instancia: Plataforma
+ getInstancia(): Plataforma
+ obtenerVideosByCanal(c: Canal): Contenido[*]
+ obtenerVideosByDirector(d: Persona): Contenido[*]
+ obtenerDuracion(c: Contenido): double
+ registrarUsuario(nick: String, clave: String, nombre: String): Usuario
- videos
- contenidos
Persona
- nombre: String
- nacionalidad: String
- fechaNacimiento: Date
- fechaDefuncion: Date[0..1]
«abstract»
Contenido
- titulo: String
- fechaCreacion: Date
- descripcion: String
getDuracion(): double
- etiquetas
«enum»
Etiquetas
+ VIOLENTO
+ DOCUMENTAL
+ ANIMACION
+ BLANCO_NEGRO
- actores
- directores
1..*
«abstract»
Reproducible
- duracion: double
+ getDuracion(): double
Serie
+ getDuracion(): double
Temporada
- numeroTemporada: int
+ getDuracion(): double
- temporadas
1..*
Pelicula
- edadMinimaRecomendada: int
Capitulo
- numeroCapitulo: int
- capitulos
1..*
«implements»
«interface»
Comparable
«implements»
suscribirse(Canal c) {
c.attach(this);
this.subjects.add(c);
}
update() {
System.out.println("Nuevo contenido");
}
@Override
addContenido(Contenido c) {
this.videos.add(c);
this.notifyPadres();
}
getParent() {
return this.parent;
}
setParent(c: CanalCompuesto) {
this.parent = c;
}
addContenido(Contenido c) {
return;
}
attach(Usuario u) {
this.observers.add(u);
}
notify() {
for (Usuario u : this.observers) {
u.update();
}
}
notifyPadres() {
this.notify();
if (this.getParent() != null) {
this.getParent().notifyPadres();
}
}
addHijo(Canal c) {
return;
}
@Override
addHijo(Canal c) {
this.subcanales.add(c);
}
registrarUsuario(String nick, String clave, String nombre) {
return new Usuario(nombre, nick, clave);
}

Para este diseño, volvemos a usar Singleton para la clase principal, en este caso representa la Plataforma de streaming, esta clase se compone de Canales y Contenidos para obtener la mayor cantidad de datos posible de la forma más eficiente (p.e. la colección de contenidos de un determinado director), ya que, al tener una lista de estos elementos, su filtrado y obtención es muy sencillo, por otro lado, maneja una lista de Usuarios para poder darles de alta en la plataforma y gestionar sus credenciales y autenticado.

Usamos Composite para la jerarquía de los canales, añadimos también algunos de sus métodos y atributos básicos de cara a facilitar las notificaciones de observer, por ejemplo, la clase padre almacena datos de su padre y los getters y setters correspondientes, también añadimos el método addHijo para crear esa jerarquía. Implementamos el método getVideos() para obtener todos los videos asociados a esa estructura, por ejemplo, en canalcompuesto este método devuelve la unión de la invocación de getVideos() de cada uno de sus subcanales.

Como se ha mencionado antes, se hace uso del patrón Observer, sin clases abstractas ni interfaces, ya que, en este concreto ejemplo, no nos va a hacer falta tener observadores y/o observados de distinto tipo. Usamos alguno de sus métodos característicos para permitir esa suscripción a un canal y las notificaciones cuando se añade el contenido, en concreto, implementamos registrarUsuario para crear observadores, suscribirse para invocar al método attach del subject, update para mostrar la notificación recibida, notify para invocar al método update de cada observador y creamos uno nuevo, notifyPadres, para permitir que los padres de las hojas avisen a sus observadores también de que una de sus hojas se ha modificado, este método, llama a su método notify asociado y si tiene un padre, invoca al del padre también.

Por último, los contenidos tienen asociados 2 atributos hacia persona en relación 1-N representando directores y actores, al no tener ninguna funcionalidad diferencial entre ellos, he decidido no implementar una herencia y 2 clases distintas. También tiene una relación 0-N con las etiquetas, puede no tener ninguna o tenerlas todas. Como capítulos y películas tienen una duración se implementa la clase abstracta Reproducible para no repetir código, al igual que Serie y Temporada, hereda de contenido. Las series se componen de por lo menos una temporada, como necesitamos que mantenga el orden, se crea un atributo numeroTemporada y hacemos que la clase implemente la interfaz pre-definida Comparable, de esta forma, llamando a Collections.sort() tendremos la lista de temporadas siempre ordenada. Lo mismo sucede con los capítulos, una temporada tiene al menos uno y sigue la misma implementación usada para las temporadas. El método getDuracion en contenido es abstracto y nos permite tener diversas implementaciones, por ejemplo, en elementos de Reproducible, devolverá esta duración, en series y temporadas, la suma de los resultados de las invocaciones sobre sus elementos, es decir, getDuracion en una serie invocará a getDuración de cada una de sus temporadas, estas, invocarán al mismo método sobre sus capítulos que devolveran, 1 a 1, su duración, la suma de estos resultados es el resultado final.