

I/O RECORDS

MEMORIA



ÍNDICE

- 1. Estructura del proyecto**
- 2. Análisis, diseño e implementación**
 - 2.1 Análisis**
 - 2.2 Diseño**
 - 2.3 Implementación**
- 3. Arquitectura y patrones**

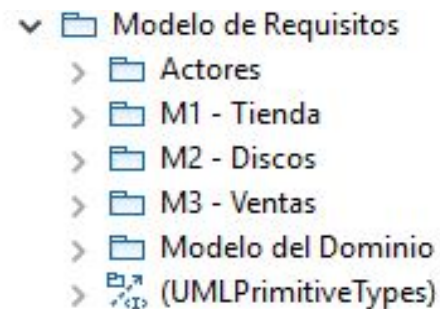
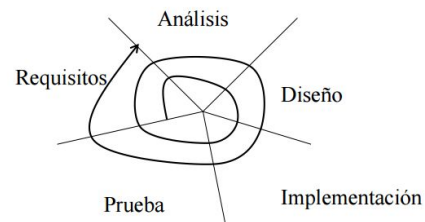
I/O RECORDS

1. Estructura del proyecto

El proceso que se sigue en el proyecto es el **proceso unificado de desarrollo** el cual sigue una estructura que está formada por cinco flujos de trabajo que se iteran: requisitos, análisis, diseño, implementación, pruebas.

De esta manera, el proyecto está organizado en tres paquetes:

En primer lugar, el **modelo de requisitos** el cual consta de los siguientes pasos (no hace falta que se realicen por separado): enumerar los requisitos candidatos, comprender el contexto del sistema, capturar los requisitos funcionales, capturar los requisitos no funcionales. Se divide en tres paquetes (*ver imagen 1*):



-**Modelo del dominio:** el objetivo de este modelo es ayudar a comprender y describir las clases más importantes dentro del contexto del sistema que se trata, por ello se proporciona un vocabulario común a clientes y desarrolladores facilitando la comprensión.

El modelo del dominio contiene un diagrama de clases en el que se indica las relaciones que existen entre los distintos módulos. Cada clase u objetos del dominio representan cosas o eventos que ocurren durante el proceso de desarrollo y contiene atributos que le caracterizan.

-**Un paquete** en el que aparecen los actores de todos los casos de uso, en nuestro caso, los actores son empleado, personal y usuario.

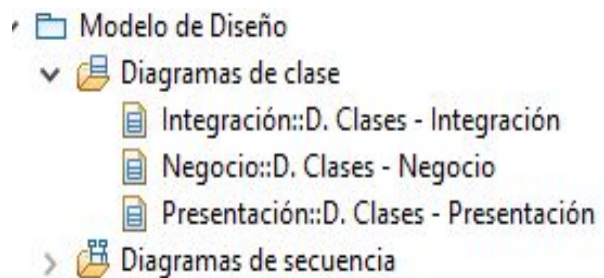
-**Un paquete por cada módulo:** M1-Tienda, M2-Discos y M3-Ventas y cada uno de estos está dividido en requisitos funcionales que contiene los casos de uso que a su vez estos incluyen los diagramas de actividad. Tanto los diagramas de casos de uso (ayudan a capturar los requisitos funcionales y hacer hincapié en cada usuario individual) y de actividad (describen el caso de uso correspondiente) hacen de guía para el desarrollo del proyecto. Los requisitos funcionales (según el módulo) presentes en nuestro proyecto son:

- M1: añadir producto, darse de alta, darse de baja, devolver producto, modificar producto, quitar producto, adquirir producto, ver catálogo, ver datos

personales y ver estadísticas.

- M2: descatalogar discos, establecer ofertas, modificar discos, ver catálogo y ver disco.
- M3: añadir pedido, eliminar pedido, generar factura, modificar pedido y ver beneficio.

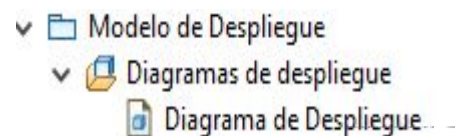
En segundo lugar, el **modelo de diseño** que es un modelo de objetos el cual describe la realización física de los casos de uso centrándose en los requisitos del sistema obtenidos anteriormente.



Está estructurado en dos paquetes (ver imagen 2), diagrama de clases y de secuencia. Además, siguiendo una arquitectura multicapa cada uno de ellos está formado por tres capas siendo estas:

- Presentación: hace referencia a la interfaz gráfica o textual. Se le atribuye la lógica para que los clientes puedan acceder al sistema.
- Negocio: Realización de acciones a base de la información recopilada desde la capa de presentación y desde la capa de integración. Esta capa también proporciona los servicios del sistema.
- Integración: hace referencia a los recursos y sistemas externos que guarda el programa.

Por último, el **modelo de despliegue** que es un modelo de objetos que sigue una distribución física de cómo funciona la conexión entre los nodos. Cada nodo representa un recurso del sistema que normalmente es un procesador o dispositivo hardware.



Dicho modelo está formado por un *diagrama de despliegue* (tipo de diagrama del Lenguaje Unificado de Modelado, ver imagen 3), así como los entornos de ejecución (hardware utilizado) y artefactos. Estos, representan elementos concretos en el mundo físico que son el resultado de un proceso de desarrollo. Ejemplos de artefactos son archivos ejecutables, bibliotecas, archivos, esquemas de bases de datos, archivos de configuración, etc. En el diagrama aparece como componente UML 1.X).

De esta manera, existe una correspondencia entre la arquitectura software y la arquitectura del sistema (el hardware).

La manera de poder observar lo anteriormente citado es gracias a la *vista arquitectónica del modelo de despliegue*, que muestra los artefactos más importantes de dicho modelo para su arquitectura.

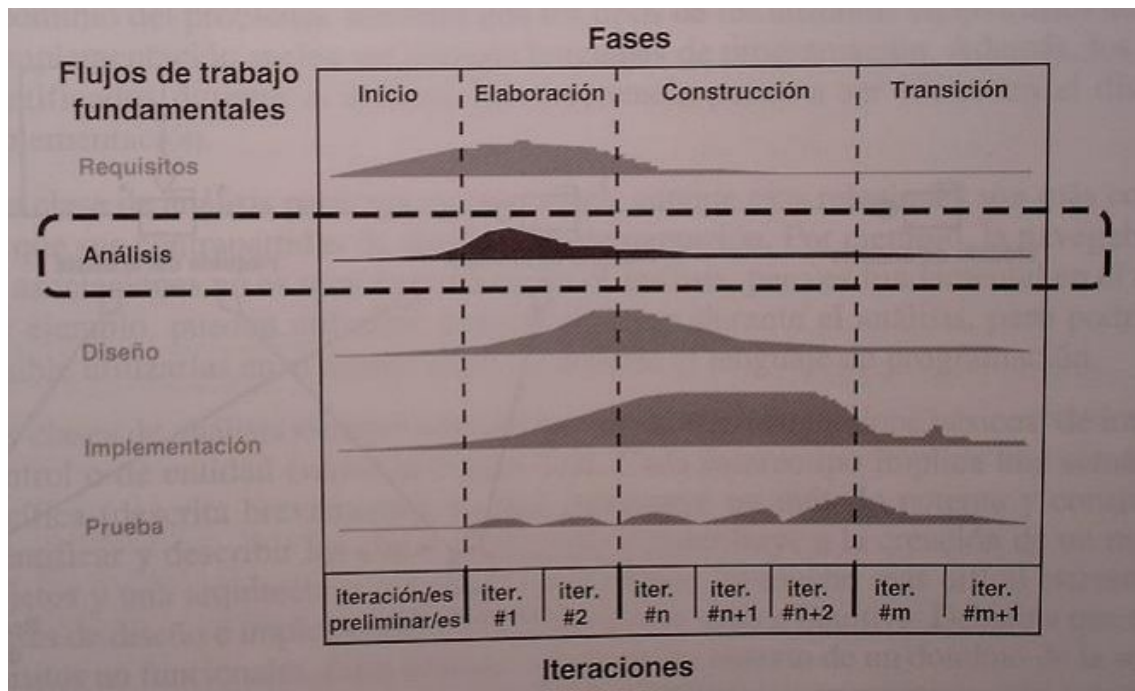
2. Análisis, diseño e implementación:

2.1 Análisis

El análisis según el Manual de Referencias de UML, es la etapa de un sistema que captura los requisitos y el dominio del problema.

Se centra en lo que hay que hacer, es decir, se escribe y se detalla los requisitos pedido. En esta fase, los desarrolladores estudian los requisitos para poder dar forma el modelo de análisis (refinado y estructurado) que cuyo objetivo es conseguir una comprensión más precisa y una descripción de los mismos que sea fácil de mantener y que ayude a estructurar el sistema (como en el diseño).

Como se puede observar en la imagen siguiente, el análisis cobra mayor importancia en la fase de elaboración y principios de la construcción que da pie a las etapas de diseño e implementación.



A continuación, se nombrará brevemente las finalidades de nuestra aplicación:

1. Facilitar el trabajo, tanto al empleado como al cliente.
2. Mejorar la interacción, consiguiendo que disminuya la tasa de errores que pueda cometer una persona, poniendo en función una aplicación.
3. Realizar multitareas para mejorar el rendimiento de la tienda.
4. Una impresión modernista.

Habiendo analizado los requisitos del proyecto (los puntos mencionados anteriormente) que se encuentran en la especificación de los requisitos se ofrece una especificación de manera más precisa a través del modelo de análisis (no incluido en el

proyecto de IBM) centralizada en el sistema y no al problema.

De esta manera, se obtiene una mayor precisión de los requisitos que el modelo de casos de uso. Es desarrollado con un lenguaje más formal para poder razonar sobre el funcionamiento interno del sistema.




El papel del análisis depende de la fase de desarrollo del software. (Ver Imagen 4). Se centra al inicio de la fase de elaboración y más adelante, al terminar la fase de elaboración y durante la construcción el énfasis, pasa al diseño que cobrará cierta importancia pues con ello, se facilitará el trabajo. Una de las ventajas que nos ofrece el análisis, es que podemos utilizarlo como una interfaz entre requisitos y diseño, ya que es similar a los al modelo de requisitos, pero con un lenguaje más formal y entendible a nivel de desarrolladores además de muy cercano al modelo de diseño.

El artefacto utilizado para capturar el análisis es el modelo de análisis, formado por: clases de análisis, realizaciones de casos de uso, paquetes de análisis y vista arquitectónica del modelo de análisis.

Por lo que respecta a una clase de análisis, es capaz de representar de forma abstracta una o varias clases con características como la centralización del tratamiento de los requisitos funcionales (detallados en la especificación de requisitos).

También se define a los atributos en una clase de análisis, aunque sean a un nivel bastante alto. Estos atributos son fácilmente reconocibles en el dominio del problema que pon el contrario, en la etapa de diseño son tipos. Estos atributos podrían a pasar a ser clases durante la etapa de diseño.

Las clases de análisis están estereotipadas en tres tipos:

- Clase de interfaz: 
- Clase de control: 
- Clase de entidad: 

Las clases de interfaz se utilizan para modelar la interacción entre el sistema y sus actores, recibir información y peticiones hacia los usuarios y sistemas externos. Además, no es necesario que describan cómo se ejecuta físicamente la interacción.

Las clases de entidad se utilizan para modelar información que posee una vida larga, es decir, una persona, un objeto del mundo real, o un suceso del mundo real.

Las clases de control representan coordinación secuencia, transacciones y control de otros objetos. Sirve para encapsular el control de un caso de uso en concreto, que a su vez puede servir para la encapsulación de procesamientos complejos pero que no pueden asociarse con una clase entidad concreta.

2.2 Diseño

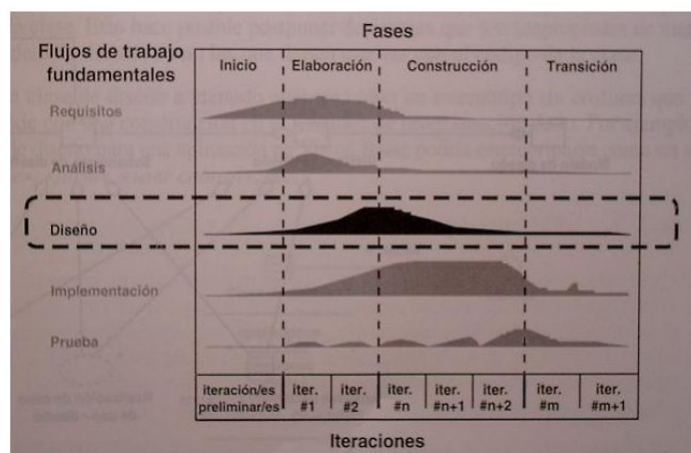
Se trata de un modelo físico cuya finalidad es dar forma al sistema intentando conservando en su mayoría las estructuras definidas en el modelo de análisis. Además, sirve de plano a la hora de realizar la implementación.

La entrada fundamental del diseño es el modelo de análisis, que proporciona una comprensión detallada y formal de los requisitos y proporciona una estructura al sistema.

Los objetivos son:

- Comprender con mayor profundidad los aspectos que están relacionados con los requisitos no funcionales y restricciones técnicas que se han detallado en el Plan de Proyecto.
- Crear una entrada apropiada y servir como punto de partida para el siguiente flujo, la implementación.
- Poder ser capaces de descomponer las diferentes partes del trabajo para que pueda facilitar a los desarrolladores el trabajo en grupo, teniendo en cuenta la concurrencia.

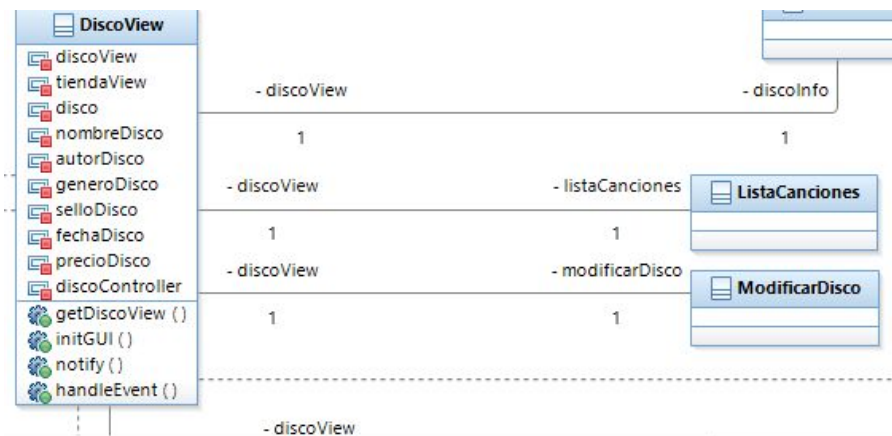
Con lo que se refiere al diseño, se le da mayor importancia a final de la fase de elaboración y a comienzos de la construcción (ver imagen 4). Esto es un paso hacia una arquitectura más sólida, estable y elaborada. Por tanto, cuando la arquitectura es más estable y todos los requisitos están claros, comienza la implementación.



El artefacto que se utiliza para capturar el diseño es el modelo de diseño que está formado por: clases de diseño, realización de caso de uso-diseño, subsistema de diseño, Interfaces, vista arquitectónica del modelo de diseño, modelo de despliegue, vista arquitectónica del modelo de despliegue.

Las clases de diseño tienen como características: El lenguaje que se utiliza coincide con el de implementación, normalmente va acompañado de atributos, las relaciones entre clases de diseño se traducen directamente en implementación, los métodos tienen correspondencia con el código, una clase de diseño puede posponer decisiones que son inapropiadas de manejar en el modelo de diseño, una clase de diseño puede realizar interfaces si tiene sentido hacerlo en el lenguaje de programación.

Todas estas características se pueden observar en el modelo de diseño. Por ejemplo, el caso de la tienda de discos:

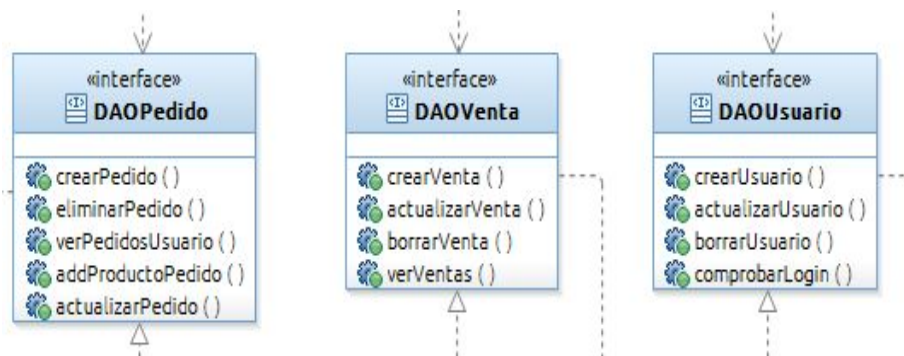


Se trata de una captura de una parte de la capa de presentación incluida en el paquete de diagrama de clases. Se puede observar que existe una asociación bidireccional entre las clases DiscoView con *ListaCanciones* y *ModificarDisco*. Además, algunas de las clases van acompañadas de atributos que la caracterizan y de operaciones.

Además de los artefactos, los subsistemas de diseño ayudan a organizarlos en piezas más manejables. Un subsistema puede constar de: clases de diseño, realizaciones de caso de uso, interfaces y otros subsistemas. Estos deben ser cohesivos y débilmente acoplados.

Los subsistemas pueden representar componentes que proporcionan varios interfaces compuestos a partir de otros, como los que especifican clases de implementación individuales, y que se convierten en ejecutables o ficheros binarios.

Por otro lado, al referirnos a interfaces se debe tener en cuenta que, si una clase de diseño realiza una interfaz, se debe incluir los métodos que realicen las operaciones necesarias. Por el contrario, si un subsistema realiza una interfaz este debe contener una clase de diseño u otros subsistemas. Este es un ejemplo de interfaces en la tienda de discos:



Por tanto, las interfaces ayudan a separar lo que refiere a las operaciones de los métodos.

Lo anteriormente relatado se puede observar mediante el *diagrama de clases de diseño*.

Además del modelo de diseño, **el modelo de despliegue** es útil pues otro tipo de artefacto o producto de trabajo que muestra la configuración de los nodos de proceso en el tiempo de ejecución, los enlaces de comunicación entre ellos, y las instancias de componente y los objetos que residen en ellas.

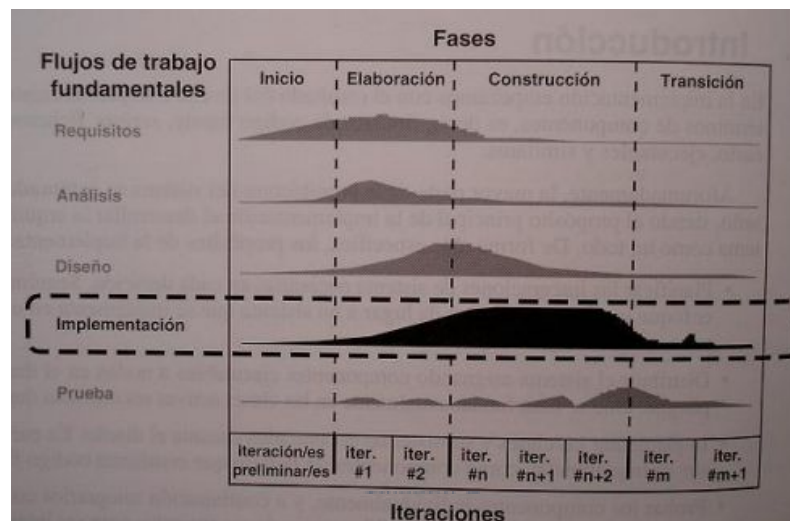
2.3 Implementación

Comienza con el resultado del diseño y se implementa el sistema en componentes, es decir, ficheros de código fuente, scripts, ficheros de código binario, ejecutables y similares.

El objetivo de este flujo se trata de:

- Definir la organización del código
- Implementar clases y objetos en forma de componentes (fuente, ejecutables, etc.) encontrados durante la etapa anterior de diseño.
- Probar las componentes desarrolladas individualmente
- Integrar las componentes en un sistema ejecutable
- Distribuir el sistema asignando componentes ejecutables a nodos en el diagrama de despliegue.

La implementación cobra mayor importancia en la fase de construcción (ver imagen 5). Como se puede observar, durante la fase de elaboración se crea la línea de base ejecutable de la arquitectura y durante la fase de transición puede tratar defectos que se localizan más tarde.



Puesto que esta etapa corresponde con el estado actual del sistema en cuanto a componentes y subsistemas se refiere, es normal mantener este estado durante el ciclo de vida del software.

De esta manera, el artefacto es **el modelo de implementación** (no incluido en el proyecto IBM), formado por: artefactos (componentes UML 1.x), subsistemas de

implementación, interfaces, visión arquitectónica del modelo de implementación, plan de integración de construcciones.

El modelo de implementación, al igual que el modelo de diseño, las clases, se implementan en términos de artefactos, como ficheros de código fuente, ejecutables, etc. Describe la organización de los artefactos y sus dependencias.

Algunos estereotipos de componentes son: **executable** (es un programa ejecutable en un nodo), **file** (es un fichero que contiene código fuente o datos), **library** (es una librería estática o dinámica), **table** (tabla de una base de datos) y **document** (documento).

Al igual que en el modelo de diseño, las clases pueden ir acompañadas de subsistemas que se manifiestan a través de un mecanismo de empaquetamiento concreto en un entorno de implementación determinado: un paquete Java, un proyecto Visual Basic, un directorio de ficheros en un proyecto C++, un paquete en una herramienta Rational Rose...

Las interfaces del modelo de implementación se corresponden con las del modelo de diseño.

A través del **plan de integración de construcciones** se describe la secuencia de construcciones necesarias en una iteración (puede haber más de una iteración por construcción), mejor dicho, la funcionalidad que se espera de la construcción y las partes del modelo de implementación afectadas por la construcción. Por lo tanto, una construcción es una versión ejecutable del sistema.

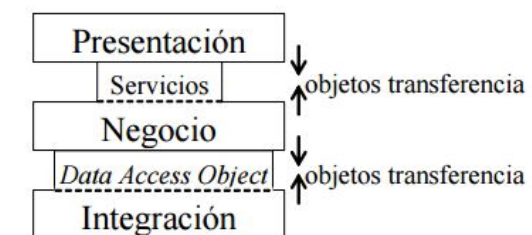
La última etapa del flujo es la **prueba**, donde se verifica el resultado de la implementación probando cada construcción, así como las versiones finales del sistema.

3. Arquitectura

Este proyecto, está basado en una arquitectura multicapa que es un conjunto ordenado de subsistemas, cada uno de los cuales están constituidos en términos de los que tiene por debajo y proporciona la base de la implementación de aquellos que están por encima de él. Los objetos de cada capa suelen ser independientes, aunque suelen haber dependencias entre objetos de distintas capas. Además, existe una relación cliente/servidor entre las capas inferiores, que son las que proporcionan los servicios, y las capas superiores, los usuarios de estos servicios.

La arquitectura multicapa dispone de tres tipos de nodos:

- Clientes que interactúan con los usuarios finales.
- Servidores de aplicación que



Arquitectura multicapa

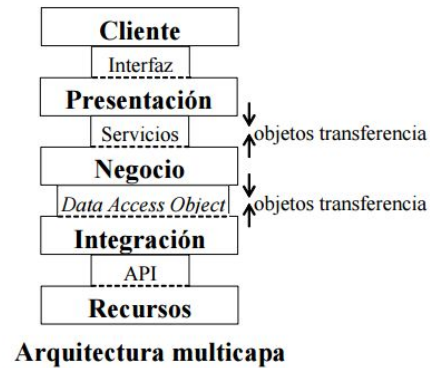
procesan los datos para los clientes.

- Servidores de la base de datos que almacenan los datos para los servidores de aplicación.

Considera una capa de presentación, otra de negocio, y otra de integración (ver imagen). Aunque en realidad, está formado por 5 capas:

- La capa de clientes representa a todos los clientes que acceden al sistema
- La capa de recursos contiene los datos del negocio y recursos externos.

Una de las ventajas de esta arquitectura es que se puede modificar cualquier capa sin afectar al resto y como desventaja la complejidad arquitectónica. Además de otras más características:



- **Centralización del control:** los accesos, recursos e integridad de los datos son controlados por el servidor, de forma que un programa cliente defectuoso o no autorizado no pueda dañar el sistema
- **Escalabilidad:** se puede aumentar la capacidad de clientes y servidores por separado. Cualquier elemento puede ser aumentado (o mejorado) en cualquier momento, o se pueden añadir nuevos nodos a la red (clientes y/o servidores).
- **Fácil mantenimiento:** al estar distribuidas las funciones y responsabilidades entre varios ordenadores independientes, es posible reemplazar, reparar, actualizar, o incluso trasladar un servidor, mientras que sus clientes no se verán afectados por ese cambio (o se afectarán mínimamente). Esta independencia de los cambios también se conoce como *encapsulación*.

Patrones utilizados

- **Singleton:** este patrón es utilizado en la creación de las factorías abstracta, controlador y en distintas clases de la capa de presentación.
- **Transferencia:** este patrón lo utilizamos en las clases básicas que forman la aplicación, como puede ser Usuario y Disco.
- **Data Access Object (DAO):** este patrón es utilizado para las clase principales para generar una conexión con la DB (utilización en la capa de integración), además en estas se implementarán patrones como singleton y factoría abstracta.

- **Servicio de aplicación (SA):** este patrón, al igual que el anterior, se usa en las clases principales, y está en la capa de negocio, donde genera una encapsulación, además en estas se implementarán patrones como singleton y factoría abstracta.

- **Modelo Vista-Controlador (MVC) y Observador:**
 - Los componentes que forman la vista son, por ejemplo, *PanelView* (es un panel en el cual solo el empleado del negocio puede acceder a este), *BarraLateral* (es un panel donde viene la información del disco, y donde se puede generar un filtro de discos), *CarritoView* (es un panel donde se muestra los discos que se han añadido a este, para ser procesados en un nuevo pedido), *CatalogoDiscos* (es un panel donde se muestran los distintos discos que se encuentran en la DB), *DiscoView* (es un panel donde se mostrará la información del disco), *LoginView* (es un panel donde se iniciará sesión con una cuenta que este establecida en la DB)...

 - El patrón se ha implementado utilizando el patrón observador-observable para comunicar el modelo con la vista. El modelo manda un objeto Notificación a la vistas que encapsula toda la información que necesita representar.

 - Quienes desempeñan el papel de modelo son *DAODisco*, *SADisco*, *DAOUsuario*, *SAUsuario*... estos son implementados a través de los patrones DAO y SA, que a su vez usa los patrones de factoría abstracta y singleton. Se encargan de encapsular y sacar la información necesaria de la DB.

- **Factoría abstracta:** este patrón es utilizado en las DAO y SA. Genera una interfaz de familia de objetos.