

**Administrador Digital de Venta de Multimedia:**

**Disc Store Manager**

Informe del Proyecto Final de las asignaturas:

Diseño y Programación Orientada a Objetos

Y Diseño de Interfaces y Pruebas.

Enunciado #11

**Autores:**

Luis Miguel Lagos Xenes

Carlos Daniel Robaina Rivero

**Facultad de Ingeniería Informática**

**La Habana, 18 de junio de 2022**

#### Resumen

La automatización de procesos, tanto a nivel de empresa, como de país, es uno de los aspectos esenciales dentro del desarrollo económico de la sociedad. Es por ello que la creación de una herramienta que permita mejorar y hacer más efectivas un determinado número de acciones tanto en un departamento empresarial, como en una tienda de venta de cualquier índole, puede resultar necesario. Tomando esto como punto de partida, el objetivo del programa que nos propusimos crear es facilitar la compra y venta de discos en una tienda de CD y DVD, así como permitir una interacción de los trabajadores de la tienda con las diferentes opciones del programa.

Las ventajas que ofrece la herramienta, van desde el registro de la venta de los discos por parte de los trabajadores, el control de los productos hasta el despido de los mismos trabajadores. O sea, que contiene un grupo amplio de funcionalidades que dinamizan las interacciones en este ámbito de la tienda y mejoran la eficiencia del negocio que se desea automatizar.

**Palabras Claves:**

**Administración**

**Tienda de discos**

**Automatización**

**Programación orientada a objetos**

Índice de Contenido

[Introducción 4](#_Toc107175295)

[Enunciado 4](#_Toc107175296)

[Objetivo del Proyecto 5](#_Toc107175297)

[Plan de Trabajo 6](#_Toc107175298)

[Estructura del Documento 9](#_Toc107175299)

[Desarrollo 10](#_Toc107175300)

[Tarjetas CRC 10](#_Toc107175301)

[Diagrama de Clases UML 14](#_Toc107175302)

[Consideraciones para la Interfaz Gráfica de Usuario 15](#_Toc107175303)

[Reportes Implementados 15](#_Toc107175304)

[Patrones de Diseño 17](#_Toc107175305)

[Patrón Builder 17](#_Toc107175306)

[Patrón Factory Method 19](#_Toc107175307)

[Patrón Proxy 20](#_Toc107175308)

[Mecanismos de Validación y Excepciones 21](#_Toc107175309)

[Pruebas Realizadas 23](#_Toc107175310)

[Pruebas de Caja Blanca 23](#_Toc107175311)

[Pruebas de Caja Negra 25](#_Toc107175312)

[Conclusiones 29](#_Toc107175313)

[Recomendaciones 31](#_Toc107175314)

[Bibliografía 32](#_Toc107175315)

# Introducción

## Enunciado

Se desea automatizar la gestión de una tienda de venta de discos. En ella se comercializa discografía de varios sellos musicales en los soportes de CDs y DVDs. Además, en la tienda se desea brindar un servicio de personalización de grabaciones según los deseos de los clientes.

De la tienda se conoce su nombre, dirección, teléfono, nombre y apellidos del gerente y fecha desde que ocupa el cargo.

Para cada trabajador de la tienda se conoce su nombre y apellidos, número de identidad, número del trabajador, que es único y no puede ser modificado, salario básico, nivel escolar y cargo que ocupa.

De cada canción se conoce el título, género, duración, autor, intérprete, interpretes colaboradores, álbum, tamaño del fichero.

De cada video clip se conoce el título, género, duración, intérprete, interpretes colaboradores, tamaño del fichero y resolución.

Se conoce el precio de cada producto musical. Por otra parte, el precio de los productos del servicio de grabación personalizado tiene un valor básico que depende del soporte donde se realice la grabación y a este valor se le añade un costo que depende de la cantidad de datos grabados.

## Objetivo del Proyecto

El proyecto que se documenta intenta ser una forma útil y eficaz de automatizar determinados procesos involucrados en una tienda de CDs y DVDs.

La principal tarea del programa realizado es la de manejar la venta de los productos, desde escoger las canciones o videos que se grabaran en el disco hasta el control de los distintos discos vendidos y qué trabajador llevó a cabo la operación. Esta funcionalidad si bien es la más importante no es la única utilidad del software ya que este pretende ser un sistema integrado lo más completo posible. Es por esto que se presentan la posibilidad de llevar el control de la multimedia disponible para ofertarle al cliente.

Además, como herramientas administrativas a disposición del gerente del negocio el programa brinda la capacidad de almacenar información de sus trabajadores y proveer una forma simple para contratar y despedir a los mismo desde la propia aplicación.

Por último, se ofrece un sistema de control de acceso al software para mantener un estricto registro del trabajador que se encuentra en su sesión de trabajo, las ventas que este realiza y permite también manejar los permisos de acceso a las funcionalidades de la aplicación en base a este sistema de autentificación.

## Plan de Trabajo

Para organizar el trabajo y optimizar el tiempo de los desarrolladores involucrados se decidió separar el proceso de trabajo en NUMERO etapas bien definidas:

* 1ra Etapa: Definir las funcionalidades necesarias para satisfacer las necesidades del público objetivo del software.
* 2da Etapa: Diseñar la primera propuesta de la capa lógica para darle solución a la problemática.
* 3ra Etapa: Diseñar las interfaces que servirán como intermediario entre el cliente de la aplicación y la capa lógica de la etapa anterior.
* 4ta Etapa: Implementar las funcionalidades diseñadas en la 2da Etapa.
* 5ta Etapa: Crear las interfaces documentadas en la 3ra Etapa y establecer el vínculo entre estas y la capa lógica.
* 6ta Etapa: Pulir e implementar los detalles del diseño del sistema e incorporar patrones de diseño.
* 7ma Etapa: Realizar pruebas de calidad de software.

|  |  |  |  |
| --- | --- | --- | --- |
| **Plan de Trabajo** | | | |
| **No.** | **Tareas** | **Plazo de Tiempo** | **Encargado** |
| **1** | Leer e interpretar el enunciado buscando los puntos más importantes, posibles clases o reportes. | **Etapa 1**  Desde: 25 de abril  Hasta: 1 de mayo | Luis M. Lagos  Carlos D. Robaina |
| **2** | Identificar las funcionalidades necesarias para cumplir las necesidades del cliente. | **Etapa 1**  Desde: 25 de abril  Hasta: 1 de mayo | Luis M. Lagos  Carlos D. Robaina |
| **3** | Identificar las clases necesarias para representar la lógica del negocio. | **Etapa 2**  Desde: 1 de mayo  Hasta: 8 de mayo | Luis M. Lagos  Carlos D. Robaina |
| **4** | Construcción de las tarjetas CRC | **Etapa 2**  Desde: 1 de mayo  Hasta: 8 de mayo | Luis M. Lagos  Carlos D. Robaina |
| **5** | Modelado de la situación utilizando UML | **Etapa 2**  Desde: 1 de mayo  Hasta: 8 de mayo | Luis M. Lagos  Carlos D. Robaina |
| **6** | Definir directrices para la construcción de las interfaces gráficas de usuario. | **Etapa 3**  Desde: 9 de mayo  Hasta: 15 de mayo | Luis M. Lagos  Carlos D. Robaina |
| **7** | Construcción de las interfaces gráficas de usuario. | **Etapa 3**  Desde: 9 de mayo  Hasta: 15 de mayo | Luis M. Lagos  Carlos D. Robaina |
| **8** | Trasladar a Java las clases identificadas tomando como base el Diagrama UML | **Etapa 4**  Desde: 16 de mayo  Hasta: 29 de mayo | Luis M. Lagos  Carlos D. Robaina |
| **9** | Implementar sistema de autentificación al programa | **Etapa 4**  Desde: 16 de mayo  Hasta: 29 de mayo | Carlos D. Robaina |
| **10** | Implementar sección de ventas de CDs y DVDs. | **Etapa 4**  Desde: 16 de mayo  Hasta: 29 de mayo | Luis M. Lagos |
| **11** | Implementar funcionalidades del registro de productos | **Etapa 4**  Desde: 16 de mayo  Hasta: 29 de mayo | Carlos D. Robaina |
| **12** | Implementar funcionalidades de Recursos Humanos: contratar y despedir trabajadores | **Etapa 4**  Desde: 16 de mayo  Hasta: 29 de mayo | Luis M. Lagos |
| **13** | Crear ventanas autenticación y manejo de registro de productos. | **Etapa 5**  Desde: 30 de mayo  Hasta: 12 de junio | Carlos D. Robaina |
| **14** | Crear ventanas de Recursos Humanos y de ventas. | **Etapa 5** Desde: 30 de mayo  Hasta: 12 de junio | Luis M. Lagos |
| **15** | Establecer vínculos entre las interfaces y la capa lógica del sistema. | **Etapa 5**  Desde: 30 de mayo  Hasta: 12 de junio | Luis M. Lagos  Carlos D. Robaina |
| **16** | Análisis de las particularidades de la problemática y la utilización de patrones de diseño | **Etapa 6**  Desde: 13 de junio  Hasta: 19 de junio | Luis M. Lagos  Carlos D. Robaina |
| **17** | Adaptación de los prototipos de los patrones de diseño a la situación real que se pretende resolver | **Etapa 6**  Desde: 13 de junio  Hasta: 19 de junio | Luis M. Lagos  Carlos D. Robaina |
| **18** | Rectificación de errores identificados en las funcionalidades implementadas. | **Etapa 6**  Desde: 13 de junio  Hasta: 19 de junio | Luis M. Lagos  Carlos D. Robaina |
| **19** | Realizar pruebas a las interfaces y procesos lógicos de las funcionalidades de autentificación y registro de productos. | **Etapa 7**  Desde: 20 de junio  Hasta: 25 de junio | Luis M. Lagos |
| **20** | Realizar pruebas a las interfaces y procesos lógicos de las funcionalidades de venta de productos y control de trabajadores. | **Etapa 7**  Desde: 20 de junio  Hasta: 25 de junio | Carlos D. Robaina |

## 

## Estructura del Documento

El documento que se presenta tiene una estructura dividida en introducción, desarrollo y conclusiones. Así mismo el cuerpo de este se encuentra separado en una especie de epígrafes y subepígrafes que delimitan los principales temas a tratar, desde los detalles del diseño hasta las pruebas de calidad de software realizadas. Para más información consultar el [Índice de Contenidos.](#Indice)

# Desarrollo

El cuerpo de desarrollo de este informe se compone de varias secciones que pretenden dar un orden lógico al proceso de documentación sobre la construcción de software.

## Tarjetas CRC

Las Tarjetas CRC son una herramienta de diseño de software orientado a objetos usada principalmente como primer paso del proceso de modelado de la solución informática a desarrollar.

|  |  |
| --- | --- |
| Store | |
| Contener toda la información necesaria para el funcionamiento de los distintos controladores que se encargan de las funcionalidades de la propia tienda. | AccessController, DBController, HRController, SalesController |

|  |  |
| --- | --- |
| Worker | |
| Contener la información relativa a cada trabajador |  |
| Comprobar la coincidencia de sus propias credenciales con las credenciales introducidas por el usuario. | Credential |
| Actualizar sus credenciales en caso de cambios de contraseña. | Credential |

|  |  |
| --- | --- |
| Product | |
| Contener la información general de cada producto de la tienda. |  |

|  |  |
| --- | --- |
| Song | |
| Contener la información específica de los productos del tipo canción |  |

|  |  |
| --- | --- |
| Video | |
| Contener la información específica de los productos del tipo video |  |

|  |  |
| --- | --- |
| Disk | |
| Crear una abstracción que unifique las características de DVDs y CDs. |  |

|  |  |
| --- | --- |
| DVD | |
| Ser un contenedor de productos del tipo video. | Video |
| Agregar y eliminar videos de su propia lista. | Video |
| Calcular su costo total |  |

|  |  |
| --- | --- |
| CD | |
| Ser un contenedor de productos del tipo canción | Song |
| Agregar y eliminar canciones de su propia lista. | Song |
| Calcular su costo total |  |

|  |  |
| --- | --- |
| AccessController | |
| Comprobar si el trabajo que inicia sesión es el administrador | Worker |
| Autenticar un usuario en el sistema. | Worker |
| Comprobar si es la primera vez que el usuario inicia sesión | Worker |

|  |  |
| --- | --- |
| SalesController | |
| Crear el disco a petición del cliente | CDManager, DVDManager |
| Manejar la venta a un cliente | SCManager |

|  |  |
| --- | --- |
| HRController | |
| Cambiar el administrador de la tienda | Worker |
| Contratar trabajadores | Worker |
| Despedir trabajador | Worker |

|  |  |
| --- | --- |
| DBController | |
| Añadir y eliminar productos a la tienda. | Song, Video, Songbuilder, Videobuilder |
| Registrar resoluciones permitidas para los videos | Resolution |

|  |  |
| --- | --- |
| SearchManager | |
| Comprobar si existe un producto en la base de datos. | Song, Video |

|  |  |
| --- | --- |
| CDManager | |
| Ser un intermediario entre el SalesController y un CD | CD, SalesController |

|  |  |
| --- | --- |
| DVDManager | |
| Ser un intermediario entre el SalesController y un DVD | DVD, SalesController |

|  |  |
| --- | --- |
| ShoppingCard | |
| Ser un contener de discos | Disk |
| Agregar y eliminar discos de su propia lista | Disk |
| Calcular su costo total |  |

|  |  |
| --- | --- |
| SCManager | |
| Ser un intermediario entre el SalesController y el ShoppingCard | ShoppingCard |
| Generar reporte de ventas | SellReports |

|  |  |
| --- | --- |
| SellReport | |
| Guardar la información de una venta y el trabajador que la realizó | Disk, Worker |

|  |  |
| --- | --- |
| Address | |
| Contener la información organizada de una dirección |  |

|  |  |
| --- | --- |
| Credential | |
| Contener la información de inicio de sesión de un trabajador |  |

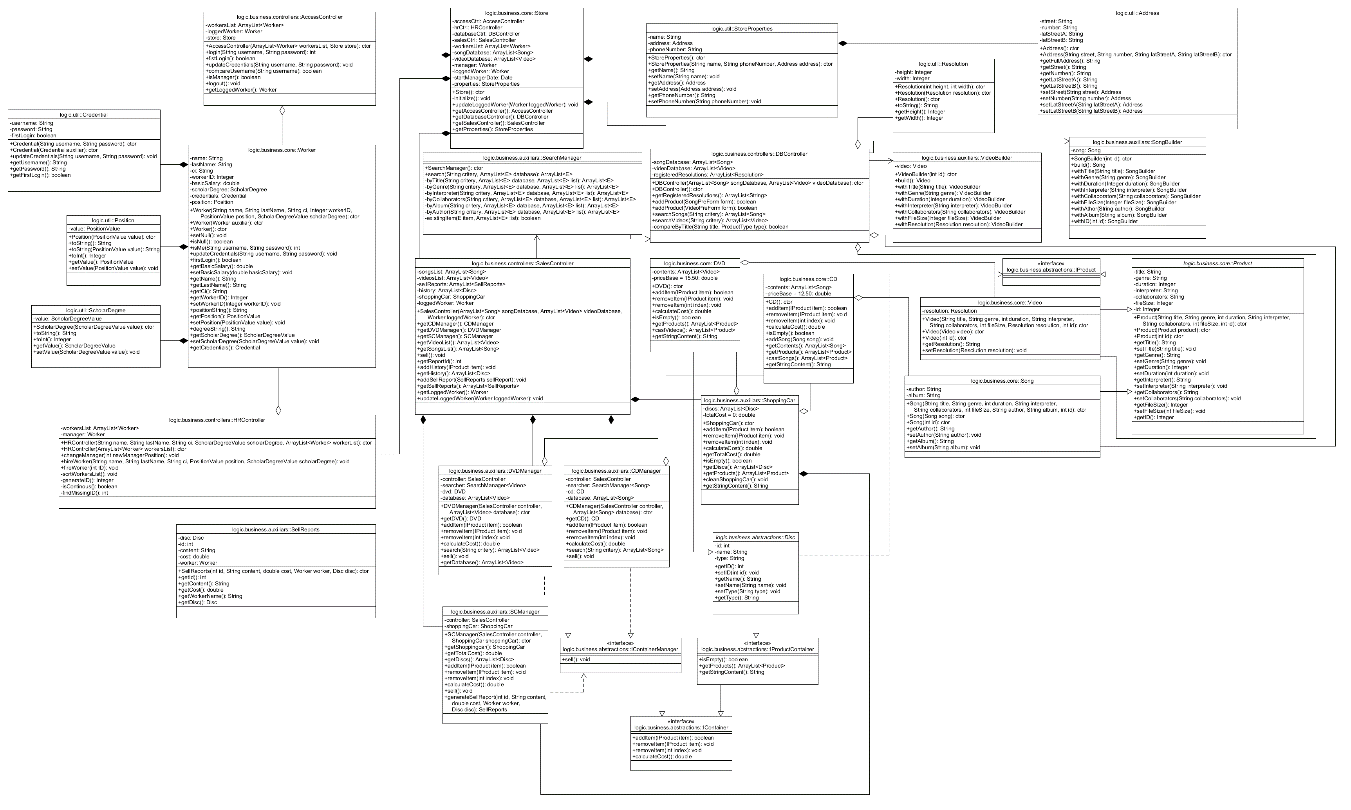
|  |  |
| --- | --- |
| StoreProperties | |
| Contener una abstracción de la información de la tienda. | Address |

|  |  |
| --- | --- |
| Validator | |
| Validar la información introducida por el usuario. |  |

## 

## Diagrama de Clases UML

El Diagrama de Clases UML permite modelar la capa lógica del software a realizar planetando las clases a implementar, sus atributos y métodos, así como las relaciones entre ellas.



Esta imagen es solo como referencia y tener constancia de su existencia en el documento oficial del proyecto. Para una mejor comprensión el Diagrama de Clases será incluido como un archivo externo en formato .png, permitiendo así un análisis correcto del mismo.

## Consideraciones para la Interfaz Gráfica de Usuario

A continuación se plantean una serie de consideraciones y pautas que se siguieron durante la contrucción de las interfaces gráficas de usuario.

Para empezar se eligió un diseño simple para la interfaz, sin elementos estéticos pretenciosos y haciendo uso del color de forma mínima. El diseño se apega lo suficiente al modelo general del sistema operativo, manteniendo sus mismos preceptos.

El software presentado hace uso de distintos tipos de elementos visuales, desde simples botones hasta complejas tablas de contenido. Siempre tartando de escoger el elemento que mejor cumpla con el objetivo deseado, valorando el tipo de dato a mostrar y la forma en la que este se mostrara. La interfaz es cien por ciento amigable con el usuario, poniendo como principal prioridad la comprensión de esta y su uso intuitivo, además de una curva de aprendizaje suficientemente llana.

El usuario se encuentra en control del flujo de acción todo el tiempo, sabiendo facilmente qué proceso se encuentra ejeuctando y cómo salir de este, ya sea cancelandolo o terminandolo satisfactoriamente. Del mismo modo la memoria de la persona que usa el programa no se encuentra sobrecargada en lo más mínimo. Esto es logrado con una combinación de intuitividad y una fácil comprensión de los controles como botones o pestañas, teniendo cada una de estas un cartel indicativo lo sufcientemente explicito.

Por último se muestra una interfaz consistente, sin muchos cambios de distribución, mensajes para nada ambiguos y perfectamente comprensibles. Los elementos mantienen su posición en la pantalla, siempre utilizando un tamaño adecuado y estándar para todo el sistema

## Reportes Implementados

Para empezar se debe aclarar que los reportes han sido escogidos de manera que sean, dentro de las posibilidades, lo más cercano a la situación real y más alejado del ámbito académico. Por esta razón los reportes, en muchos casos, combinan los 2 criterios pedidos para calificarlos.

Como primer reporte implentado se presenta el sistema de autentificación de usuarios. Este subsistema del proyecto permite la identificación del trabajador que está en su turno de trabajo y que por tanto está usando el programa en ese momento. En primera instancia dicho procedimiento debe analizar las lista de trabajadores dentro de la clase Store(Tienda), usando el AccessController como mediador y controlador, el atributo Credentials(Credenciales) de cada uno de los Worker(Trabajador) y chequear la coincidencia o no del usuario y la contraseña introducida con las credenciales de algún trabajador. Esta propuesta en si contiene 3 reportes individuales: Usuario Incorrecto, Contraseña Incorrecta y Autentificación Correcta. En este caso podemos identificar el trabajo con las lista de trabajadores existente en la tienda(workersList) y el procesamiento de la información de las clases Store y Worker, pertenecientes a la capa de la lógica del negocio y la clase Credential pertenciente a la capa de lógica auxiliar.

Este primer reporte es extremadamente útil no solo para esta situación específica, sino para muchas variantes de sistemas informáticos. Este te permite llevar un control y registro de los trabajadores que han estado utliziando la aplicación. Pero lo mejor es que este registro puede luego servir como base para próximas funcionalidades o pequeñas mejorías que se le pueden realizar al software poco a poco. Funcionalidades como llevar un reporte de las ventas por trabajador y conocer el trabajador que más aporta a la economía del negocio son solo algunas ideas del potencial que tiene esta idea.

Como próximo reporte implementado aparece la existencia del llamado “Carrito de Compras”. Este elemento le permite al usuario llevar la cuenta de los productos que está encargando el cliente, la cantidad, el tipo y el precio son algunas de las informaciones brindadas por este sistema. La capacidad de manipular los productos permite poder eliminiar productos que ya no son deseados por el cliente de la tienda, así como conocer de forma rápida el costo de su compra.

Este reporte implementa principalemente trabajo con listas, en un primer nivel una lista de los discos que desea comprar el usuaio. Agregar más discos, quitarlos, consultar su información y su precio son las herramientas brindadas por esta subsistema. Al mismo tiempo existe un trabajo con listas en un nivel inferior, que vendrían siendo las listas de productos que contienen cada disco en su interior. Permitiendo acciones y funcionalidades muy similares a las del “Carrito de Compras”.

## Patrones de Diseño

Los patrones de diseño son unas técnicas utilizadas para resolver problemas comunes en el desarrollo de sistema de software y otros ámbitos referentes al diseño de interacción o interfaces. [[1]](#Uno)

Después de realizar un análisis de la problemática que sea desea resolver con este proyecto se identificaron distintas situaciones que requerían una solución un poco más compleja que el simple análisis de los desarrolladores. En base a esto, se llevó a cabo una investigación de los patrones de diseño más usados y que mejor se pudieran adaptar a esta situación particular. A continuación, se describen detalladamente los patrones que se decidieron utilizar y la forma en que se implementaron para resolver el problema.

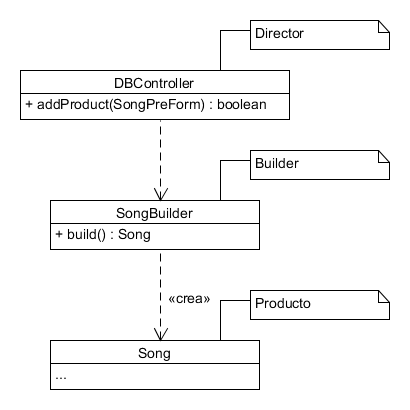
### Patrón Builder

Se usa para permitir la creación de una variedad de objetos complejos desde un objeto fuente, el cual se compone de una diversidad de partes que contribuyen individualmente a la creación de cada objeto complejo a través de un conjunto de llamadas secuenciales a una implementación especifica. Así, cada implementación existente construirá un objeto complejo de una forma diferente deseada.

En la situación particular que se plantea en el proyecto, no existe la necesidad de definir varias implementaciones distintas de un mismo objeto complejo. Sin embargo, si existe la necesidad de simplificar la creación de un objeto complejo para evitar errores y escribir un código más legible y mantenible.

A partir de las clases Song y Video se instancian objetos de una relativa complejidad debido a la cantidad de atributos de estas. Por esta razón, la llamada a un constructor de estas clases podía resultar ambigua y de difícil comprensión. Es precisamente para esto, que se aplica el patrón Builder.

Este patrón, en este caso, permite separar la construcción de un objeto de estas clases, párametro a parámetro conformando el objeto paso a paso. Esta implementación se realiza a partir de dos clases auxiliares: SongBuilder y VideoBuilder. Estas reciben las directivas de creación del objeto y al final devuelven una instancia de la clase deseada con todos los requerimientos necesarios y libre de errores. En este caso, la clase que requiere de la utilización de este sistema es la clase DBController, quien le indica al SongBuilder o VideoBuilder, los parámetros que debe tener el objeto a crear.



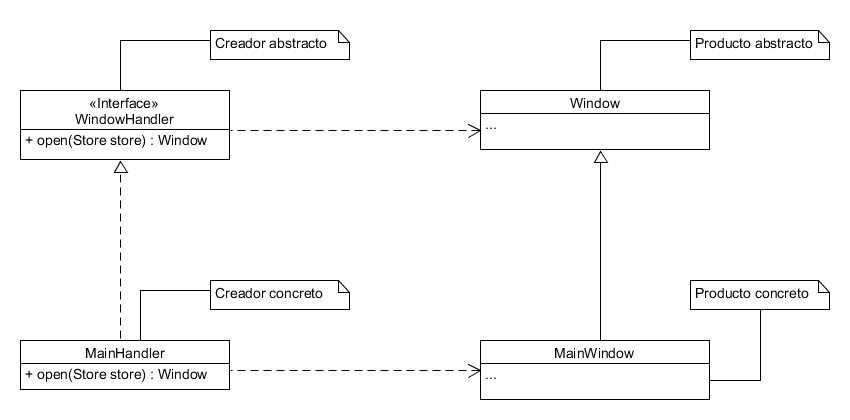
### Patrón Factory Method

 El patrón de diseño Factory Method consiste en definir una interfaz para la creación de un objeto, pero permite que las subclases decidan cuál de de las clases instanciar.

En esta situación particular se utiliza para instanciar las ventanas de la interfaz gráfica. La clase principal Application manda a instanciar una ventana, sin embargo es la subclase de la clase fábrica quién sabe exactamente que ventana concreta instanciar.

Este método aparece debido a la necesidad de encontrar una forma de crear eficientemente las interfaces gráficas, evitando el tener que pasarle como parámetro a cada ventana toda la información de la aplicación. Para esto se implementó un diseño que permite controlar la instanciación de ventanas desde la clase principal que se encarga de controlar el flujo del programa, en este caso la clase Application. Sin embargo, este método implicaba una gran cantidad de condicionales para tratar cada ventana además de la dificultad a la hora de añadir una nueva ventana.

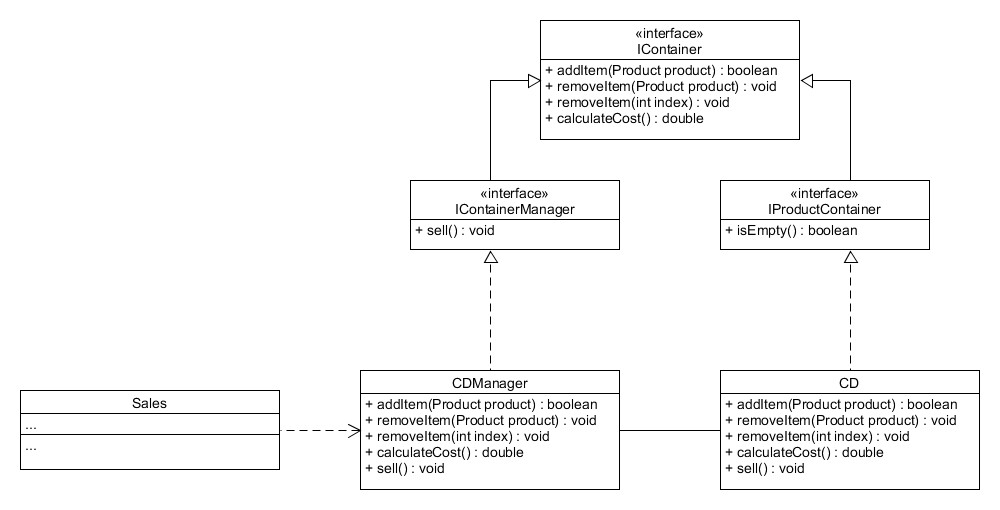
Al aplicar el patrón Factory Method esta tarea se simplifica muchísimo ya que la clase cliente, Application, no necesita conocer que ventana específica está creando. En su lugar se utiliza el polimorfismo para tratar a todos las ventanas y sus creadores como un solo tipo de objeto, delegando a cada subclase creadora la tarea de implementar la ventana correspodiente.



### Patrón Proxy

El patrón Proxy brinda un sustiituto de otro objeto con el interés de controlar el acceso al mismo. Esto puede ser ya sea por razones de problemas de desempeño o por razones de seguridad de acceso al objeto en cuestión.

La situación donde fué identificada la necesidad de utilizar dicho patrón de diseño se trata de la creación y el manejo de los distintos contenedores de productos, ya sean CDs, DVDs o el Carrito de Compra. Con vista a separar la implementación lógica de la interfaz gráfica se hacía necesario evitar que la ventana de compras interactuara directamente con los contenedores de productos, los productos dentro de estos y el registro de todos los productos de la tienda. Así mismo, también era una buena idea no delegar todas estas responsabilidades en la clase controladora del módulo de ventas, el SalesController. Por estas 2 razones se decide implementar las clases CDManager, DVDManager y SCManager, las cuales actúan como intermediario, o proxy, entre la ventana y el conteneder de los productos, permitiendo también la modularización de la clase controladora y la división de responsabilidades.



## Mecanismos de Validación y Excepciones

En el programa se utilizaron diferentes mecanismos que funcionaron como medios para comprobar la integridad y validez de los datos que se pueden introducir y sus componentes. De esta forma garantizamos que los datos que el usuario introduzca cumplan los requisitos y los parámetros de acorde al programa.

En este sentido, trabajamos diferentes aspectos esenciales dentro de la validación, entre los que se destacan: comprobación de que los datos que introduzca el usuario se encuentren en un valor aceptable que puede ser superior o inferior dentro de un determinado rango, comprobación de que la cantidad de caracteres esté acorde a la cantidad que se pida, comprobación de que se haga efectiva la introducción de una entrada por parte del usuario y de esta forma evitando que se quede algún campo vacío (Activando un mensaje de error que impedirá seguir al siguiente paso sin rellenar el campo), comprobación de que los datos introducidos sean del tipo que se esperen (pueden ser numéricos, caracteres).

También trabajamos para asegurar que los datos sigan un orden o patrón lógico y establecido. Al mismo tiempo nos aseguramos de que el usuario solo pueda escoger una opción de las listas pre-elaboradas, de tal forma que se evitaran confusiones a la hora de comprender lo que se le pregunta o existieran errores ortográficos.

Utilizamos el método de identificador único, el comúnmente llamado ID, ya que en nuestro programa nos fue factible emplearlo para poder diferenciar los objetos (canciones, videos, discos y trabajadores) sin acceder a sus propiedades.

En el caso particular de las excepciones se utilizaron para, como se mencionó anteriormente, validar la no existencia de campos vaciós. También fueron empledas para comprobar que ciertas acciones no trabajaran con datos ilógicos, por ejemplo que los botones de borrar elementos no funcionaran a menos que existiera al menos un elemento para ser borrado. Así mismo se validó la posibilidad de existencia de referencias vacías o punteros nulos, con el objetivo de que la capa lógica tuviera su independencia de la interfaz a la hora recibir los datos correctamente.

## Pruebas Realizadas

A continuación se docuementa el diseño de las pruebas realizadas al software en cuestión, describiendo el método utilizado, los factores involucrados y por supuesto el encargado de diseñar la prueba y de implementarla.

### Pruebas de Caja Blanca

**Clase:** CD

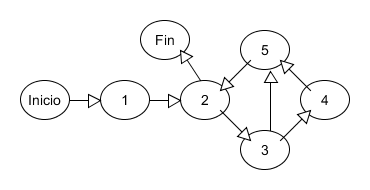
**Método:** removeItem(IProduct item)

**Código:**

|  |
| --- |
| **public** **void** removeItem (IProduct item) {  **for** (**int** i = 0; i<contents.size(); i++) {  **if**(contents.get(i).equals((Song)item)) {  contents.remove(i);  }  }  } |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Caso de Prueba | | | | | |
| Método | | Prueba de Camino Básico | | | |
| Desarrollador | | Luis Miguel Lagos Xenes | | | |
| Probador | | Carlos Daniel Robaina Rivero | | | |
| Fecha | | 26 de junio de 2022 | | | |
| Combinaciones de Variables de Entrada | | | | Resultados Esperados | Resultados Reales |
| CP | Nombres de variables de entrada | | Valor para las variables de entrada |
| 1 | item | | Objeto IProduct | No se entra al for, por tanto no se elimina ningún objeto |  |
| 2 | item | | Objeto IProduct | Nunca se encuentra el objeto, así que no se elimina ningún objeto |  |
| 3 | item | | Objeto IProduct | Se encuentra el objeto y se elimina |  |

|  |  |
| --- | --- |
| Nodo | Sentencia |
| 1 | **public** **void** removeItem (IProduct item) {  **int i = 0;** |
| 2 | **i<contents.size();** |
| 3 | **if**(contents.get(i).equals((Song)item)){ |
| 4 | **contents.remove(i);** |
| 5 | **i++** |



Camino 1: 1, 2

Camino 2: 1, 2, 3, 4, 5, 2

Camino 3: 1, 2, 3, 5, 2

Complejidad ciclomática = 3

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Caso de Prueba | | | | | |
| Método | | Prueba de Condiciones | | | |
| Desarrollador | | Luis Miguel Lagos Xenes | | | |
| Probador | | Carlos Daniel Robaina Rivero | | | |
| Fecha | | 26 de junio de 2022 | | | |
| Combinaciones de Variables de Entrada | | | | Resultados Esperados | Resultados Reales |
| CP | Nombres de variables de entrada | | Valor para las variables de entrada |
| 1 | contents.size() | | 0 | Condición del for = false  No se entra al for y por tanto no se elimina ningún objeto |  |
| 2 | contents.size() | | 6 | Condición del for = true  Se entra al for y se analiza el if interior |  |
| 3 | equals() | | false | Condición del if = false  No se entra al if y por tanto no se elimina ningún objeto |  |
| 4 | equals() | | True | Condición del if = true  Se se entra al if y se elimina el objeto deseado |  |

Condiciones: for, if

for = false -> contents.size() = 6

for = true -> contents.size() = 0

if = false -> contents.get(i).equals((Song)item) = false

if = true -> contents.get(i).equals((Song)item) = true

### Pruebas de Caja Negra

**Clase:** AccessController

**Método:** login(String username, String password)

**Código:**

|  |
| --- |
| **public** **int** login (String username, String password)  {  **int** workerID = -2;  **for** (**int** i = 0; i < workersList.size() && loggedWorker == **null**; i++)  {  Worker analizedWorker = workersList.get(i);  **int** auxiliar = analizedWorker.isMe(username, password);  **switch** (auxiliar) {  **case** 1:  workerID = -1;  **break**;  **case** 2:  workerID = analizedWorker.getWorkerID();  loggedWorker = analizedWorker;  store.updateLoggedWorker(loggedWorker);  **default**:  **break**;  }  }  **return** workerID;  } |

**Historias de Usuario:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Indentificador** | **Nombre de HU** | | | **Prioridad** | | | **Puntos Estimados** |
| HU 1 | Autentificación | | |  | | |  |
| **Usuario:** Trabajador de la Tienda | | | | | | | |
| **Descripción:** El trabajador intenta autentificarse en la aplicación | | | | | | | |
| **Programador Responsable:** Carlos Daniel Robaina Rivero | | | | | | | |
| **Escenarios** | | | | | | | |
| **Nombre** | | **Condición** | | | | **Resultado Esperado** | |
| Credenciales Incorrectas | | workersList.size() <= 0  OR  analizedWorker.isMe(username, password) = 0 | | | | No se puede acceder a la aplicación y se muestra un mensaje de error | |
| Contraseña Incorrecta | | analizedWorker.isMe(username, password) = 1 | | | | El nombre de usuario es incorrecto pero la contraseña no. | |
| Autentificación Correcta | | analizedWorker.isMe(username, password) = 2 | | | | Se accede exitosamente a la aplicación | |
| **Variables** | | | | | | | |
| **Nombre** | | | **Tipo o Dominio** | | **Descripción** | | |
| workersList.size() | | | Número entero positivo | | Cantidad de trabajadores | | |
| analizedWorker.isMe(username, password) | | | Número entero positivo | | Resultado de saber si las credenciales pertenecen a determinado trabajador | | |

**Casos de Pruebas:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Historia de Usuario** | | | HU 1: Autentificación | | | |
| **Desarrollador** | | | Carlos D. Robaina Rivero | | | |
| **Probador** | | | Luis M. Lagos Xenes | | | |
| **Fecha** | | | 25 de junio de 2022 | | | |
| **Versión** | | | 1.0 | | | |
| **Objetivo de la Prueba** | | |  | | | |
| **Descripción de la Prueba** | | |  | | | |
| **Condiciones** | | |  | | | |
| **Combinaciones de valores de entrada** | | | | | **Resultados Esperados** | **Resultados Reales** |
| **CP** | **Escenario** | **Nombres de variables** | | **Combinación de valores** |
| **1** | Credenciales Incorrectas | **[**workersList.size(),analizedWorker.isMe(username, password)**]** | | **[**3, 0**]** | No se puede acceder a la aplicación y se muestra un mensaje de error |  |
| **2** | Contraseña Incorrecta | **[**workersList.size(),analizedWorker.isMe(username, password)**]** | | **[**3, 1**]** | El nombre de usuario es incorrecto pero la contraseña no. |  |
| **3** | Autentificación Correcta | **[**workersList.size(),analizedWorker.isMe(username, password)**]** | | **[**3, 2**]** | Se accede exitosamente a la aplicación |  |

# Conclusiones

Para la solución al problema planteado se optó por desglosar el mismo en las siguientes capas:

-Gestión de Ventas

-Gestión de la Base de Datos

-Gestión de Recursos Humanos

Cada una como núcleo dentro del funcionamiento de la tienda, donde: el Gestor de Ventas se encargaría del proceso de producción del disco, la venta del mismo y la generación de un reporte a partir de la venta. El Gestor de Base de Datos se encargaría de la incorporación de nuevos productos (canciones o videos) a la tienda y el Gestor de Recursos Humanos se encargaría de toda la interacción con los trabajadores: contratar nuevos empleados, despedirlos, hacer variaciones en los salarios entre otras funciones.

Cada uno de los gestores organizado mediante el uso de patrones de diseño asegurando la eficiencia y la calidad del software, siendo algunos de estos Factory, Proxy u otros...

\*Se optó por la idea de permitir al usuario armar cada disco de forma personalizada a la hora de realizar la compra con el objetivo de tener un software más amigable y útil teniendo en cuenta su función, dejando también margen a una selección pre elaborada de títulos a elección del cliente en caso de que así desee adquirir.

\*El software posee un sistema de búsqueda en la base de datos eficiente y cómodo para realizar las tareas de búsqueda.

\*La interfaz es amigable e intuitiva, con sus respectivos bloques de información útil al usuario en algunas de las ventanas.

\*Se encuentran validados todos los campos interactivos para evitar la introducción de datos erróneos dentro del programa y con información suficiente para advertir al usuario de sus errores.

# Recomendaciones

Se recomienda para futuras actualizaciones del software:

1-La implementación de filtros de búsqueda en el proceso de conformación del disco.

2-La implementación de una interfaz para la realización de envíos a domicilio donde se calculará el costo del envío en dependencia la distancia a recorrer del mismo.

3-La implementación de un método que devuelvan listados de los productos más o menos vendidos para así hacer aún más cómoda y agradable la experiencia de usuario.

# Bibliografía

[1] E. Gamma, R. Helm, R. Johnson, J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software.* Estados Unidos: Addison-Wesley, 1994