

**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 #26

**Facultad de Ingeniería Informática**

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

# Resumen

Índice de Contenido

[Resumen 2](#_Toc106490815)

[Introducción 2](#_Toc106490816)

[Enunciado 2](#_Toc106490817)

[Objetivo del Proyecto 3](#_Toc106490818)

[Plan de Trabajo 4](#_Toc106490819)

[Desarrollo 8](#_Toc106490820)

[Tarjetas CRC 8](#_Toc106490821)

[Diagrama de Clases UML 8](#_Toc106490822)

[Consideraciones para la Interfaz Gráfica de Usuario 8](#_Toc106490823)

[Patrones de Diseño 8](#_Toc106490824)

[Mecanismos de Validación y Excepciones 8](#_Toc106490825)

[Pruebas Realizadas 8](#_Toc106490826)

[Conclusiones 9](#_Toc106490827)

[Recomendaciones 10](#_Toc106490828)

[Bibliografía 11](#_Toc106490829)

# 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 posiblidad de llevar el control de la multimedia disponible para ofertale 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 intermediarion 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** | Indentificar 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 |

# Desarrollo

## Tarjetas CRC

|  |  |
| --- | --- |
| 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

## Consideraciones para la Interfaz Gráfica de Usuario

## 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 (Design paterns. Elements of reusable object-oriented software).

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.

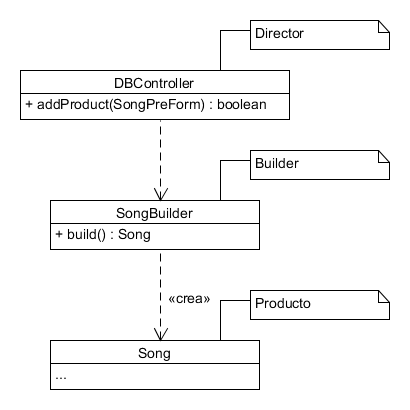
### Patrón Builder

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.

## Mecanismos de Validación y Excepciones

## Pruebas Realizadas

# Conclusiones

# Recomendaciones

# Bibliografía