Documento de diseño de Sistema

- Control de Versiones
    - Version 0.1.0 - Autores: Cavanagh Juan, Casabella Martin, Kleiner Matías - Fecha: 26/06/2017 - Descripción: primera versión del documento de arquitectura

1. Introducción

1.1 Propósito

El Documento de de Diseño de Sistema presenta el diseño de Arquitectura utilizado para el desarrollo de la aplicación BITicket, patrones de diseños aplicados, diagramas de clase, paquete, despliegue, secuencia, entre otros. Con el fin de facilitar el entendimiento del software desarrollado.

1.2 Alcance

El documento se centra en todo tipo de diagramas que permita el entendimiento del desarrollo de software. abarca los patrones de arquitectura utilizados, relaciones de clases, paquetes, componentes, etc.

1.3 Definiciones, Acrónimos y Abreviaciones

UML : Lenguaje de modelado unificado.
CSV : comma-separated values.
MVC : Patrón de arquitectura Model-View-Controller.

1.4 Objetivos y limitaciones

Se utilizó la arquitectura MVC, que utilizando 3 componentes (Vista, Modelo y Controlador) separa la lógica de la aplicación respecto a la lógica de la vista;

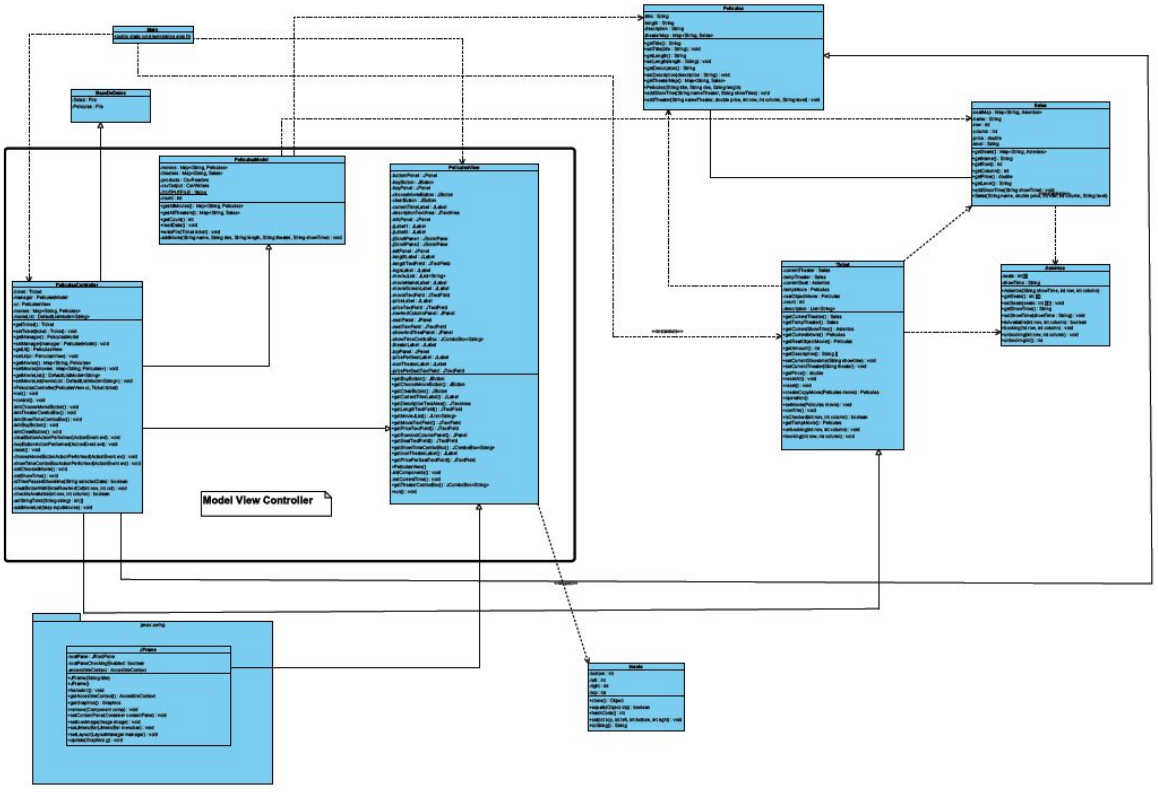
y se implementaron los patrones Observer, y Singleton.

En nuestro caso, la clase PeliculasModel, se encarga del manejo de datos, consultando a la base de datos tanto de Películas, como Salas, y actualizando constantemente, consultando, buscando, entre otras funciones.

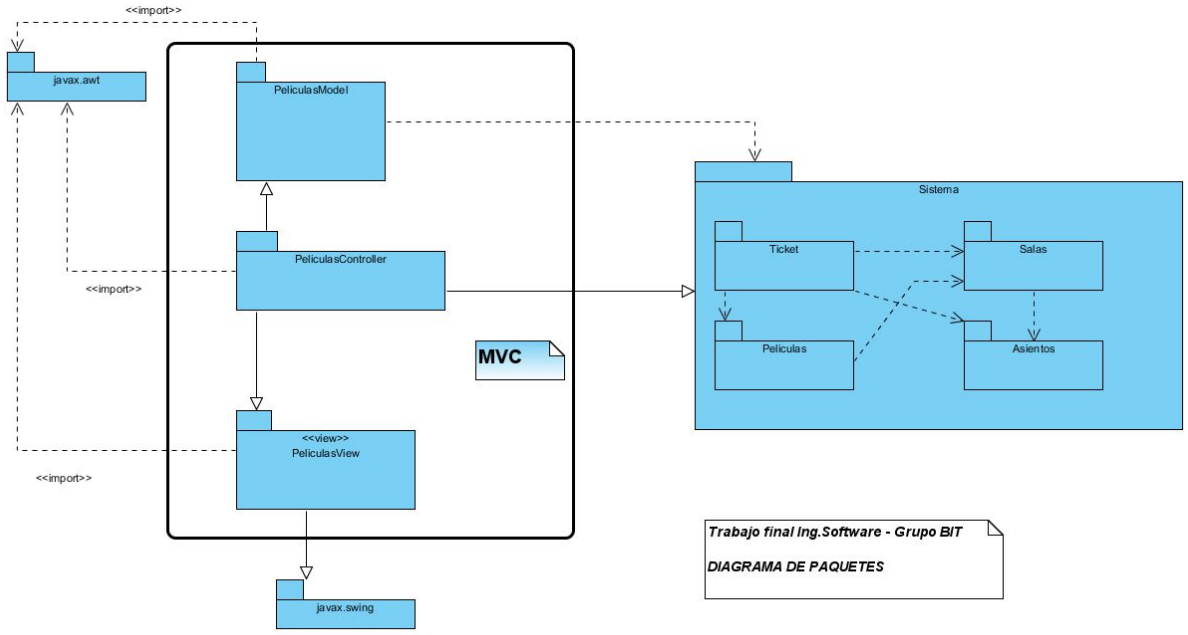
El controlador, la clase PeliculasController, recibe las órdenes del usuario y se encarga de solicitar los datos al modelo, y de comunicarlos a la vista.

PeliculasView (vista), es la representación visual de los datos, y todo lo que tenga que ver con la interfaz gráfica. Ni el modelo, ni el controlador se preocupa de cómo se verán los datos, se abstraen de dicha responsabilidad, que recae en la vista implementada.

El patrón Singleton consiste en garantizar que una clase sólo tenga una instancia y proporcionar un punto de acceso global a ella. Se implementa en PeliculasModel, permitiendo así que PeliculasController cree una instancia de PeliculasModel sólo si todavía no existe alguna, mediante el método getInstance(). Así, solucionamos el conflicto de que existan dos modelos (o más) durante la ejecución del programa.
Se procede a anexar el diagrama de clases del sistema desarrollado:



También se efectuó un diagrama de paquetes con más detalles, en donde se puede observar la relación de forma general en cuanto a las relaciones e importaciones que se realizan el sistema:

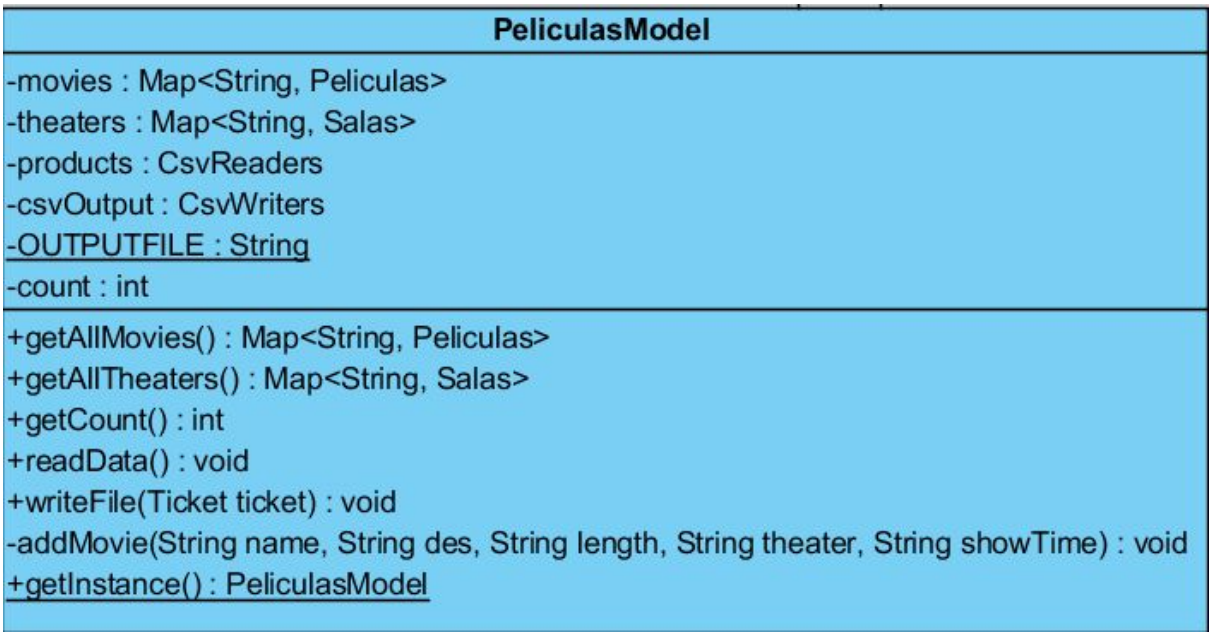


Se confeccionó también el diagrama de secuencias correspondiente al sistema, pero el mismo no se podía incluir en su totalidad en este documento.
Como los diagramas son detallados, y no se ven claramente, se anexa un link para ver ambos de forma más clara:

Link diagrama de secuencia
Link diagrama de clases
Link diagrama de paquetes

1.5 Patrones de diseño implementados

Patrón Singleton:



Como se vé en la clase, con el método getInstance, se explicita la implementación del patrón Singleton, el cual nos brinda unicidad en los datos.

Patrón Strategy:

En cuanto al patrón Strategy, él mismo se halla implementado explícitamente en la clase PeliculasController y PeliculasView, permitiendo que en un futuro, de desear presentar la misma información con diferente diseño gráfico, se debería sobrescribir o implementar nuevos métodos en la clase PeliculasView, y PeliculasController invocaría únicamente a dichos métodos.

1.6 Testing

Se anexa link que redirecciona al sitio donde se halla el documento de Testing:

Link documento de pruebas