



UNIVERSIDAD DE LAS FUERZAS ARMADAS-ESPE
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN - DCCO-SS
CARRERA DE INGENIERÍA EN TECNOLOGÍAS DE LA INFORMACIÓN



PERIODO : PREGRADO OCTUBRE 2025 - MARZO 2026
ASIGNATURA : Estructura De Datos
TEMA : Aplicación de Listas Enlazadas modelo MVC
ESTUDIANTE : Diego Montesdeoca y Angel Rodriguez
NIVEL-PARALELO - NRC : 30748
DOCENTE : Margoth Elisa Guaraca Moyota
FECHA DE ENTREGA : 11/11/2025

SANTO DOMINGO – ECUADOR

Tabla de contenido

1. Introducción	3
2. Objetivos	4
2.1. Objetivo General:.....	4
2.2. Objetivos Específicos:	4
3. Desarrollo / Marco Teórico/ Práctica	5
3.1. Lista Enlazada Simple	5
3.2. Estructuras de Pila y Cola	5
3.3. Patrón Modelo-Vista-Controlador (MVC).....	6
3.4. Estructura del Modelo.....	10
3.5. Implementación del Controlador	18
3.6. Implementación de la Vista.....	23
3.7. Funcionalidades Avanzadas de Manipulación	24
4. Conclusiones	40
5. Recomendaciones.....	41
6. Bibliografía/ Referencias.....	42

Tabla de contenido

Figura 3.1 Diagrama UML	7
Figura 3.2 Definición de la clase Movie.....	12
Figura 3.3 Metodos Getter and setter de movie.java	13
Figura 3.4 Método toString de movie.java	13
Figura 3.5 Implementación de la clase Node	14
Figura 3.6 Estructura basica de la clase Linkedlist	15
Figura 3.7 Métodos de inserción en la lista enlazada	16
Figura 3.8 Método de eliminación por título.....	17
Figura 3.9 Método de búsqueda y obtención de películas.....	18
Figura 3.10 Constructor y configuración del MovieController	19
Figura 3.11 Manejo de eventos ActionListener	19
Figura 3.12 Listeners de selección y documento.....	20
Figura 3.13 Movimiento de película al inicio	21
Figura 3.14 Movimiento de película al final	21
Figura 3.15 Validación de datos en saveMovie()	22
Figura 3.16 Actualización e inserción de películas	22
Figura 3.17 Método de eliminación de películas.....	23

1. Introducción

El presente informe detalla el desarrollo de una aplicación de software destinada a la gestión dinámica de una colección de objetos de tipo "Película". El proyecto se concibió como un ejercicio práctico en el campo de las estructuras de datos, centrándose en la implementación de una Lista Enlazada Simple para el almacenamiento y manipulación de la información, complementada con estructuras adicionales como Pila (LIFO) y Cola (FIFO) para expandir la funcionalidad del sistema.

Para asegurar una arquitectura de software robusta, escalable y modular, se adoptó el patrón de diseño Modelo-Vista-Controlador (MVC). El Modelo encapsula las estructuras de datos (Lista Enlazada Simple, Pila y Cola) y la lógica de negocio; la Vista se encarga de la interfaz gráfica de usuario (GUI); y el Controlador actúa como el puente que gestiona las interacciones del usuario y coordina las actualizaciones entre el Modelo y la Vista.

La aplicación final permite a un usuario realizar operaciones fundamentales sobre la colección de películas, tales como la inserción, eliminación, búsqueda y reorganización de elementos, demostrando la operatividad de las estructuras de datos en un entorno de desarrollo Java con interfaz gráfica, incluyendo un módulo demostrativo para visualizar las operaciones de Pila y Cola en tiempo real.

2. Objetivos

2.1. Objetivo General:

Desarrollar una aplicación de gestión de información en el lenguaje de programación Java, implementando una Lista Enlazada Simple con estructuras adicionales de Pila y Cola, utilizando el patrón MVC para la organización arquitectónica.

2.2. Objetivos Específicos:

- Implementar una lista enlazada simple en Java con todas sus operaciones básicas (inserción al inicio/final, eliminación, búsqueda y recorrido) para gestionar un catálogo de películas, complementada con estructuras de Pila y Cola para funcionalidades extendidas.
- Desarrollar una interfaz gráfica utilizando el patrón MVC que permita visualizar y manipular los datos de las películas, incluyendo funcionalidades CRUD, búsqueda en tiempo real, reorganización de elementos y un módulo demostrativo de estructuras de datos.
- Integrar mecanismos de validación de datos en el formulario de entrada para garantizar la integridad de la información, verificando campos obligatorios, formato del año y evitando duplicados, con notificaciones al usuario mediante mensajes emergentes.

3. Desarrollo / Marco Teórico/ Práctica

El proyecto se sustenta en dos pilares teóricos principales: la Lista Enlazada Simple y el patrón Modelo-Vista-Controlador (MVC).

3.1. Lista Enlazada Simple

Una lista enlazada es una colección de elementos, denominados nodos, donde el orden no está dado por su ubicación física contigua en la memoria (como en los arreglos), sino por referencias explícitas (Gómez, 2019). En una Lista Enlazada Simple, cada nodo posee dos campos: uno para almacenar la información relevante (el dato, en este caso, un objeto Pelicula) y otro campo, denominado enlace o siguiente (next), que contiene la dirección de memoria del nodo siguiente en la secuencia (Rodríguez, 2020). El último nodo de la lista tiene su campo de enlace apuntando a un valor null, lo que señala el final de la estructura. La lista se gestiona mediante una referencia principal conocida como cabeza (head), que apunta al primer nodo (Fernández, 2021).

Las Listas Enlazadas Simples ofrecen una ventaja significativa sobre las estructuras estáticas como los arrays al permitir una expansión dinámica sin requerir memoria extra para reacomodar elementos, además de facilitar la inserción y eliminación de nodos de forma más eficiente en términos de movimiento de datos (López, 2022).

3.2. Estructuras de Pila y Cola

Complementando la lista enlazada, se implementaron dos estructuras de datos adicionales:

Pila (LIFO - Last In First Out): Estructura donde el último elemento en ser insertado es el primero en ser removido (Martínez, 2020). En esta aplicación, la pila se utiliza para

almacenar el historial de películas vistas recientemente, permitiendo operaciones de apilado (push) y desapilado (pop).

Cola (FIFO - First In First Out): Estructura donde el primer elemento en ser insertado es el primero en ser removido (García, 2021). En el contexto de la aplicación, la cola gestiona una lista de películas "por ver", implementando operaciones de encolado (enqueue) y desencolado (dequeue).

3.3. Patrón Modelo-Vista-Controlador (MVC)

El patrón MVC es una arquitectura de diseño que separa la representación de la información de la interacción del usuario con dicha información, promoviendo la reutilización del código y la organización (Pérez, 2020):

Modelo (Model): Contiene la lógica del negocio y la estructura de datos (ListaEnlazada.java, Nodo.java, Pelicula.java, Pila.java, Cola.java). Es la capa responsable de gestionar el estado de la aplicación y las operaciones sobre los datos (Silva, 2022).

Vista (View): Presenta los datos al usuario y captura sus entradas (VentanaPrincipal.java, PanelFormulario.java, PanelLista.java, PanelEstado.java, DialogoDemoEstructuras.java). Es la capa de la interfaz gráfica que muestra información al usuario y recibe sus interacciones (Ramírez, 2021).

Controlador (Controller): Responde a los eventos del usuario (acciones en la Vista) y determina qué lógica del Modelo debe ejecutarse, actualizando la Vista con los resultados (ControladorPelicula.java). Actúa como intermediario entre la Vista y el Modelo, coordinando el flujo de datos (Torres, 2023).

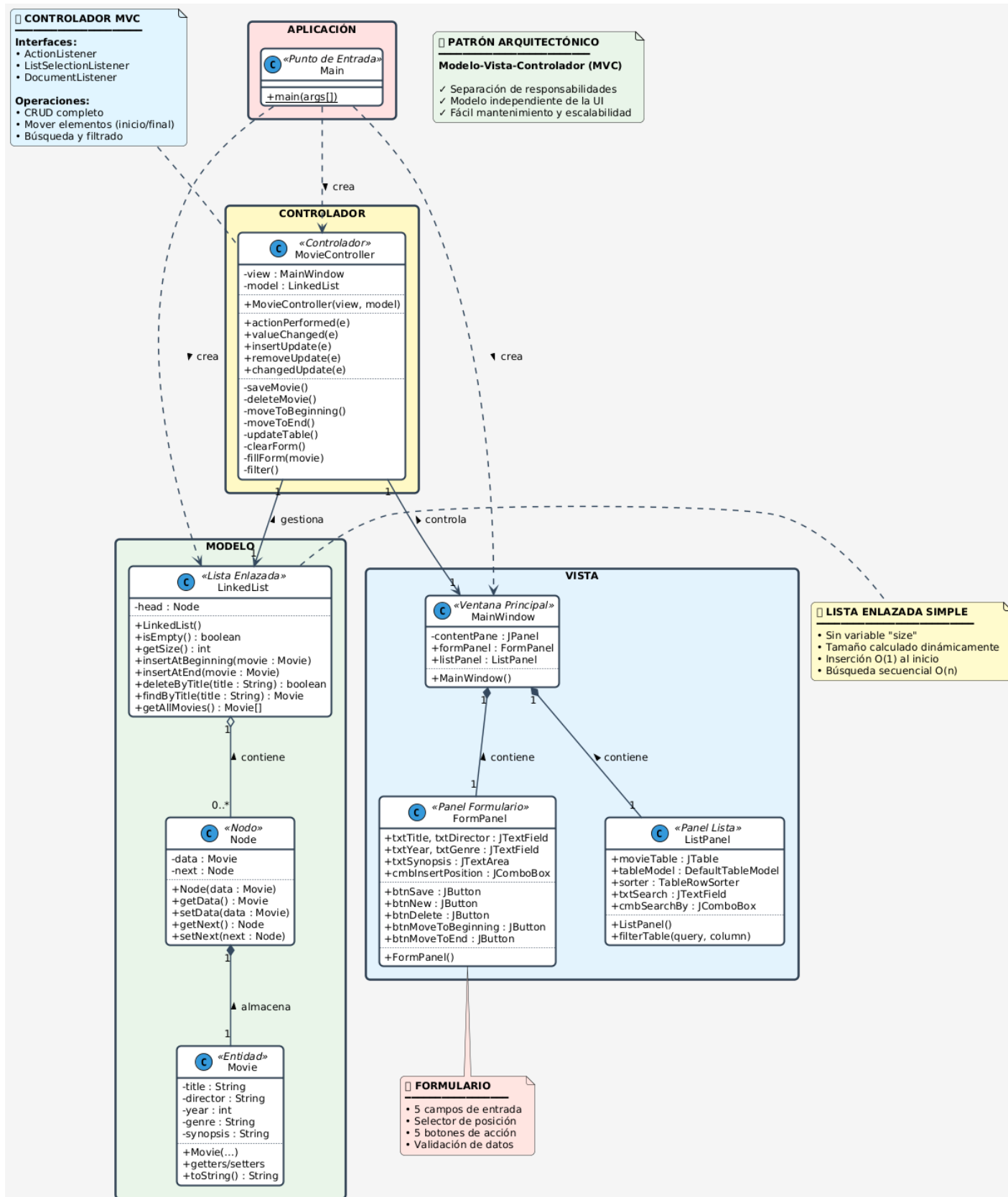
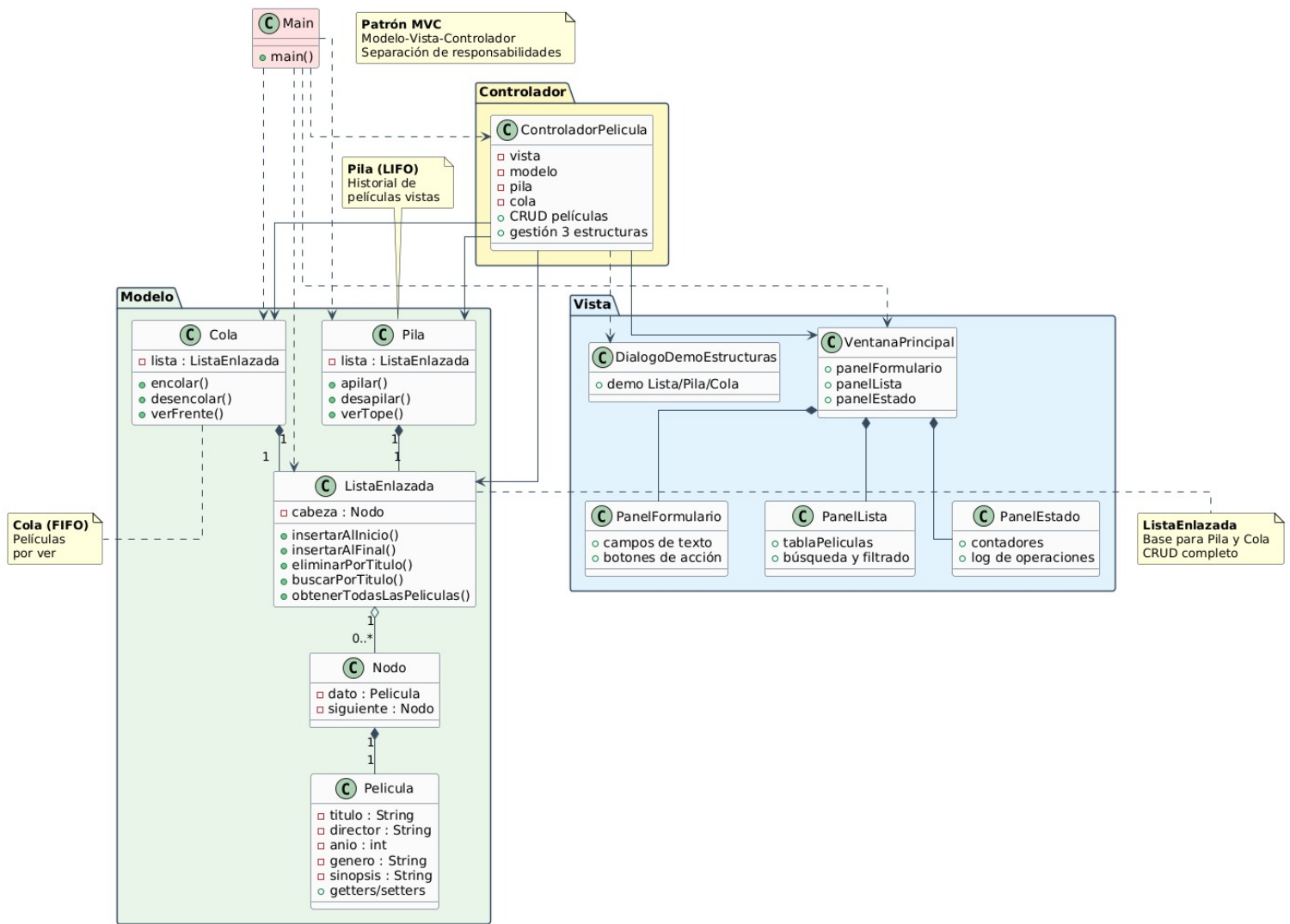
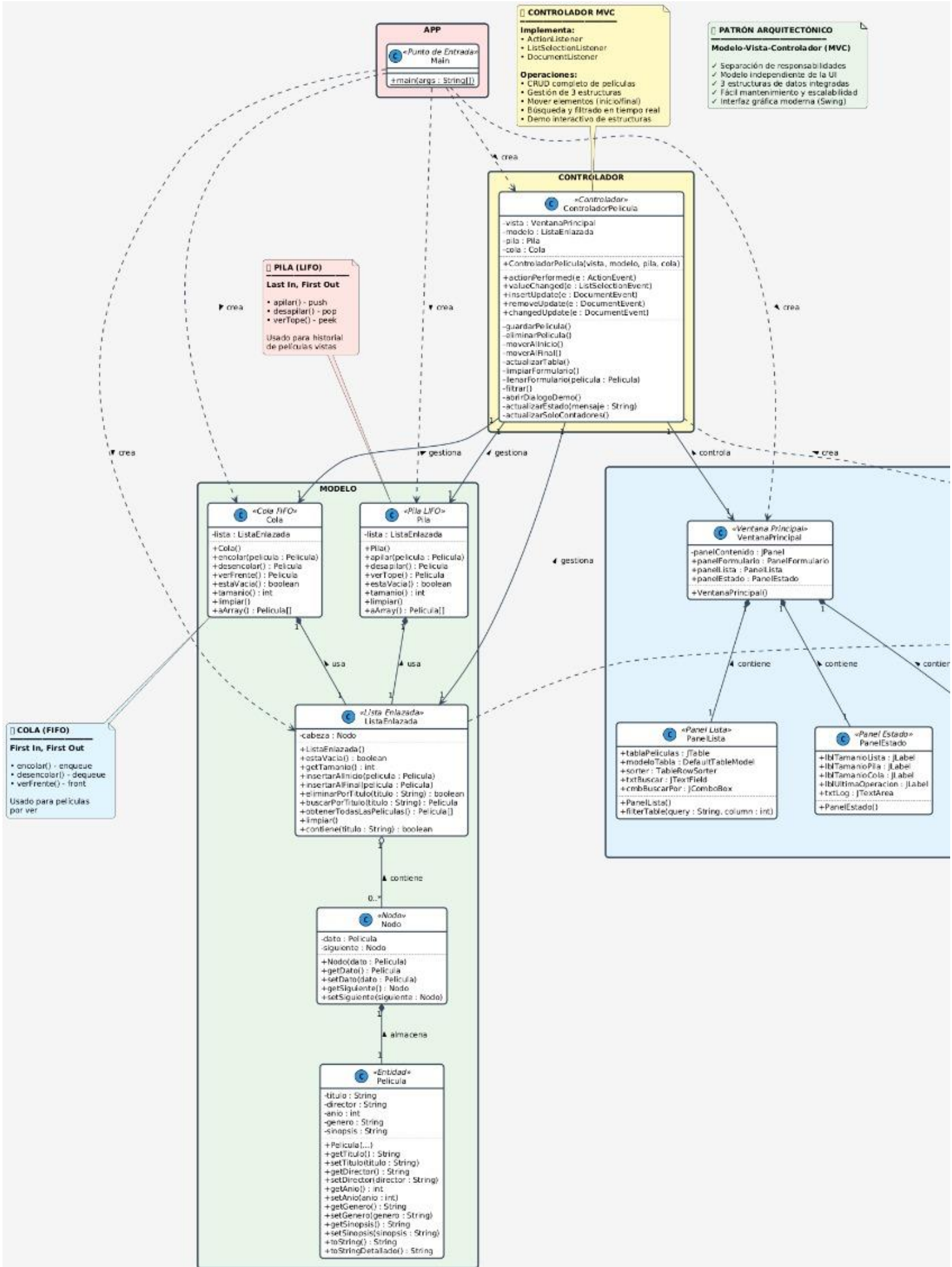
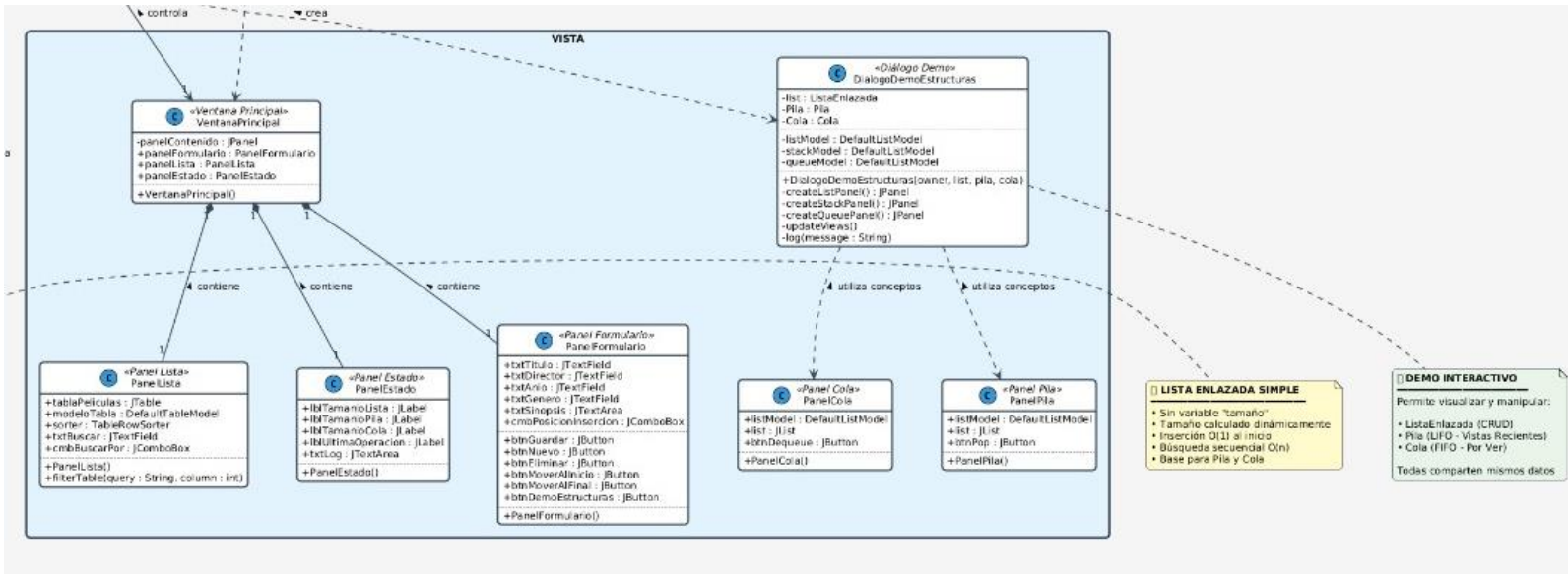


Figura 3.1 Diagrama UML







Práctica (Análisis de la Implementación)

La implementación se realizó en Eclipse Java y se estructuró en los paquetes Model, View, Controller y App, conforme a los requisitos de la práctica.

3.4. Estructura del Modelo

Pelicula.java: Se definió con atributos para almacenar el título, director, año, género y sinopsis de la película, junto con sus métodos getter y setter, y validaciones integradas para garantizar la integridad de los datos.

3.5. Pilas y Colas

Las pilas y las colas son estructuras de datos lineales que restringen la forma en que se insertan y eliminan los elementos. En una pila, se aplica el principio LIFO (Last In, First Out), donde el último elemento insertado es el primero en ser extraído. Las operaciones básicas en una pila son push (inserción) y pop (eliminación) (Martínez, 2021). Por otro lado, la cola sigue el principio FIFO (First In, First Out), en el que el primer elemento en entrar es el primero en salir, con operaciones típicas como encolar (insertar) y desencolar

(eliminar) (López, 2022). Ambas estructuras son fundamentales en la informática para resolver problemas relacionados con gestión de memoria, algoritmos de recorrido y procesos secuenciales (Fernández, 2023).

3.6. Colas

La cola es una estructura de datos lineal que sigue el principio FIFO (First In, First Out), es decir, el primer elemento en entrar es el primero en salir. La inserción de elementos se realiza en el extremo final de la cola, mientras que la eliminación se hace en el frente. Las colas se utilizan comúnmente en problemas de procesamiento en orden, como la gestión de procesos en sistemas operativos, la simulación de eventos y la administración de tareas en impresoras. Existen variaciones como colas circulares, de prioridad y dobles colas, pero la cola básica representa un modelo eficiente para el manejo secuencial de elementos (Mosquera, 2024; UETITC, 2022).

```

1 package Modelo;
2
3 import java.util.Objects;
4
5 /** Película con información básica y validación */
6 public class Pelicula {
7
8     private String titulo;
9     private String director;
10    private int anio;
11    private String genero;
12    private String sinopsis;
13
14    public Pelicula(String titulo, String director, int anio, String genero, String sinopsis) {
15        setTitulo(titulo);
16        setDirector(director);
17        setAnio(anio);
18        setGenero(genero);
19        setSinopsis(sinopsis);
20    }
21
22    public String getTitulo() {
23        return titulo;
24    }
25
26    public void setTitulo(String titulo) {
27        if (titulo == null || titulo.trim().isEmpty()) {
28            throw new IllegalArgumentException("El título no puede estar vacío");
29        }
30        this.titulo = titulo.trim();
31    }
32
33    public String getDirector() {
34        return director;
35    }
36
37    public void setDirector(String director) {
38        if (director == null || director.trim().isEmpty()) {
39            throw new IllegalArgumentException("El director no puede estar vacío");
40        }
41    }
42

```

Figura 3.2 Definición de la clase Pelicula

```

public String getTitulo() {
    return titulo;
}

public void setTitulo(String titulo) {
    if (titulo == null || titulo.trim().isEmpty()) {
        throw new IllegalArgumentException("El titulo no puede estar vacío");
    }
    this.titulo = titulo.trim();
}

public String getDirector() {
    return director;
}

public void setDirector(String director) {
    if (director == null || director.trim().isEmpty()) {
        throw new IllegalArgumentException("El director no puede estar vacío");
    }
    this.director = director.trim();
}

public int getAño() {
    return año;
}

public void setAño(int año) {
    final int AÑO_MIN = 1888;
    final int AÑO_MAX = java.time.Year.now().getValue() + 1;

    if (año < AÑO_MIN || año > AÑO_MAX) {
        throw new IllegalArgumentException("El año debe estar entre " + AÑO_MIN + " y " + AÑO_MAX);
    }
    this.año = año;
}

public String getGenero() {
    return genero;
}

public void setGenero(String genero) {
    if (genero == null || genero.trim().isEmpty()) {
        throw new IllegalArgumentException("El género no puede estar vacío");
    }
    this.genero = genero.trim();
}

public String getSinopsis() {
    return sinopsis;
}

```

Figura 3.3 Metodos Getter and setter de movie.java

```

49     @Override
50     public String toString() {
51         return this.title + " (" + this.year + ")";
52     }
53 }

```

Figura 3.4 Método toString de pelicula.java

Nodo.java: Cada nodo almacena un objeto Película y una referencia siguiente de tipo Nodo, confirmando la implementación de una Lista Enlazada Simple.

```

1 package Model;
2 public class Node {
3
4     private Movie data;
5     private Node next;
6
7     // Constructor
8     public Node(Movie data) {
9         this.data = data;
10        this.next = null;
11    }
12
13    // Getters and Setters
14    public Movie getData() {
15        return data;
16    }
17
18    public void setData(Movie data) {
19        this.data = data;
20    }
21
22    public Node getNext() {
23        return next;
24    }
25
26    public void setNext(Node next) {
27        this.next = next;
28    }
29 }

```

Figura 3.5 Implementación de la clase Node

ListaEnlazada.java: Implementó las operaciones clave de una lista enlazada:

- `getTamano()`: Retorna el número de elementos en la lista
- `insertarAlInicio(Pelicula pelicula)`: Inserta un elemento al principio de la lista
- `insertarAlFinal(Pelicula pelicula)`: Inserta un elemento al final de la lista
- `eliminarPorTitulo(String titulo)`: Elimina un elemento por su título

- `buscarPorTitulo(String titulo)`: Busca un elemento por título
- `obtenerTodasLasPeliculas()`: Retorna todos los elementos como array

Pila.java y Cola.java: Implementan las operaciones fundamentales de estas estructuras (apilar/desapilar, encolar/desencolar) con validaciones y manejo de estados.

```

1 package Model;
2 public class LinkedList {
3     private Node head;
4     public LinkedList() {
5         this.head = null;
6     }
7     public boolean isEmpty() {
8         return head == null;
9     }
10    public int getSize() {
11        int count = 0;
12        Node current = head;
13        while (current != null) {
14            count++;
15            current = current.getNext();
16        }
17        return count;
18    }

```

Figura 3.6 Estructura básica de la clase *LinkedList*

insertAtBeginning(Movie movie): La inserción requiere crear un nuevo nodo y reasignar la referencia `head` para que apunte al nuevo nodo, y el nuevo nodo apunte al nodo previamente en la cabeza.

insertAtEnd(Movie movie): Requiere recorrer la lista desde `head` hasta que el campo `next` del nodo actual sea `null`, y luego asignar el nuevo nodo a ese campo.

```

19 // Insert at the beginning of the list
20 public void insertAtBeginning(Movie movie) {
21     Node newNode = new Node(movie);
22     newNode.setNext(this.head);
23     this.head = newNode;
24 }
25 // Insert at the end of the list
26 public void insertAtEnd(Movie movie) {
27     Node newNode = new Node(movie);
28     if (isEmpty()) {
29         this.head = newNode;
30     } else {
31         Node current = this.head;
32         while (current.getNext() != null) {
33             current = current.getNext();
34         }
35         current.setNext(newNode);
36     }
37 }

```

Figura 3.7 Métodos de inserción en la lista enlazada

deleteByTitle(String title): Se debe localizar el nodo a eliminar y el nodo anterior.

Una vez localizado, se redefine el puntero next del nodo anterior para que apunte al sucesor del nodo a eliminar, excluyéndolo de la lista. Se maneja un caso especial si el nodo a eliminar es la head.

```

38 // Delete an element by its title
39 public boolean deleteByTitle(String title) {
40     if (isEmpty()) {
41         return false;
42     }
43     if (head.getData().getTitle().equalsIgnoreCase(title)) {
44         head = head.getNext();
45         return true;
46     }
47
48     Node previous = head;
49     Node current = head.getNext();
50     while (current != null && !current.getData().getTitle().equalsIgnoreCase(title)) {
51         previous = current;
52         current = current.getNext();
53     }
54     if (current != null) {
55         previous.setNext(current.getNext());
56         return true;
57     }
58     return false;
59 }

```

Figura 3.8 Método de eliminación por título

findByTitle(String title): Implica un recorrido secuencial de la lista, comparando el título del objeto Movie en cada nodo, retornando el objeto Movie si se encuentra, o null en caso contrario.

getAllMovies(): Realiza un recorrido de la lista para obtener todos los objetos Movie en un arreglo, para su posterior visualización en la tabla de la interfaz.

```

60 // Find a node by title
61 public Movie findByTitle(String title) {
62     Node current = head;
63     while (current != null) {
64         if (current.getData().getTitle().equalsIgnoreCase(title)) {
65             return current.getData();
66         }
67         current = current.getNext();
68     }
69     return null;
70 }
71 // Get all movies to display them
72 public Movie[] getAllMovies() {
73     if (isEmpty()) {
74         return new Movie[0];
75     }
76     int totalSize = getSize();
77     Movie[] movies = new Movie[totalSize];
78     Node current = head;
79     int i = 0;
80     while (current != null) {
81         movies[i] = current.getData();
82         current = current.getNext();
83         i++;
84     }
85     return movies;
86 }
87 }

```

Figura 3.9 Método de búsqueda y obtención de películas

3.7. Implementación del Controlador

El controlador incluye métodos especializados para:

Gestión de películas: guardarPelicula(), eliminarPelicula(), actualizarTabla()

Manipulación de posición: moverAlInicio(), moverAlFinal()

Búsqueda y filtrado: filtrar()

Demo de estructuras: abrirDialogoDemo()

```

1 package Controller;
2+ import Model.LinkedList;
14
15 public class MovieController implements ActionListener, ListSelectionListener, DocumentListener {
16
17     private MainWindow view;
18     private LinkedList model;
19
20 public MovieController(MainWindow view, LinkedList model) {
21     this.view = view;
22     this.model = model;
23
24     // Register listeners
25     this.view.formPanel.btnSave.addActionListener(this);
26     this.view.formPanel.btnNew.addActionListener(this);
27     this.view.formPanel.btnDelete.addActionListener(this);
28     this.view.formPanel.btnMoveToBeginning.addActionListener(this);
29     this.view.formPanel.btnMoveToEnd.addActionListener(this);
30     this.view.listPanel.movieTable.getSelectionModel().addListSelectionListener(this);
31     this.view.listPanel.txtSearch.getDocument().addDocumentListener(this);
32     this.view.listPanel.cmbSearchBy.addActionListener(this);
33
34     // Update table (will be empty at start)
35     updateTable();
36 }
--

```

Figura 3.10 Constructor y configuración del MovieController

```

38     // --- Listeners ---
39
40     @Override
41 public void actionPerformed(ActionEvent e) {
42     Object source = e.getSource();
43     if (source == view.formPanel.btnSave) {
44         saveMovie();
45     } else if (source == view.formPanel.btnNew) {
46         clearForm();
47     } else if (source == view.formPanel.btnDelete) {
48         deleteMovie();
49     } else if (source == view.formPanel.btnMoveToBeginning) {
50         moveToBeginning();
51     } else if (source == view.formPanel.btnMoveToEnd) {
52         moveToEnd();
53     } else if (source == view.listPanel.cmbSearchBy) {
54         filter();
55     }
56 }

```

Figura 3.11 Manejo de eventos ActionListener

```

58     @Override
59     public void valueChanged(ListSelectionEvent e) {
60         if (!e.getValueIsAdjusting()) {
61             int selectedRow = view.listPanel.movieTable.getSelectedRow();
62             if (selectedRow != -1) {
63                 // Convert view index to model index (in case it's filtered)
64                 int modelRow = view.listPanel.movieTable.convertRowIndexToModel(selectedRow);
65                 String title = (String) view.listPanel.tableModel.getValueAt(modelRow, 0);
66
67                 Movie movie = model.findByTitle(title);
68                 if (movie != null) {
69                     fillForm(movie);
70                 }
71             }
72         }
73     }
74
75     @Override
76     public void insertUpdate(DocumentEvent e) {
77         filter();
78     }
79
80     @Override
81     public void removeUpdate(DocumentEvent e) {
82         filter();
83     }
84
85     @Override
86     public void changedUpdate(DocumentEvent e) {
87         filter();
88     }
89

```

Figura 3.12 Listeners de selección y documento

Adicionalmente, el controlador implementa métodos especializados para la manipulación de la posición de elementos en la lista:

moveToBeginning(): Permite mover una película existente desde cualquier posición de la lista hacia el inicio. El método elimina el elemento de su posición actual utilizando `deleteByTitle()` y lo reinserta al principio mediante `insertAtBeginning()`, manteniendo la integridad de la estructura de datos.

```

243 private void moveToBeginning() {
244     int selectedRow = view.listPanel.movieTable.getSelectedRow();
245     if (selectedRow == -1) {
246         JOptionPane.showMessageDialog(view, "Please select a movie from the list to move.", "No Movie Selected", JOptionPane.WARNING_MESSAGE);
247         return;
248     }
249
250     // Get the movie title from the selected row
251     int modelRow = view.listPanel.movieTable.convertRowIndexToModel(selectedRow);
252     String title = (String) view.listPanel.tableModel.getValueAt(modelRow, 0);
253
254     // Find the movie in the list
255     Movie movie = model.findByTitle(title);
256     if (movie == null) {
257         JOptionPane.showMessageDialog(view, "Movie not found.", "Error", JOptionPane.ERROR_MESSAGE);
258         return;
259     }
260
261     // Remove from current position and insert at beginning
262     model.deleteByTitle(title);
263     model.insertAtBeginning(movie);
264
265     JOptionPane.showMessageDialog(view, "Movie moved to beginning successfully.", "Move Successful", JOptionPane.INFORMATION_MESSAGE);
266     updateTable();
267     clearForm();
268 }

```

Figura 3.13 Movimiento de película al inicio

moveToEnd(): Similar al anterior, pero mueve el elemento seleccionado hacia el final de la lista usando insertAtEnd(). Esta funcionalidad demuestra la flexibilidad de las listas enlazadas para reorganizar elementos sin necesidad de desplazar bloques grandes de memoria.

```

270 private void moveToEnd() {
271     int selectedRow = view.listPanel.movieTable.getSelectedRow();
272     if (selectedRow == -1) {
273         JOptionPane.showMessageDialog(view, "Please select a movie from the list to move.", "No Movie Selected", JOptionPane.WARNING_MESSAGE);
274         return;
275     }
276
277     // Get the movie title from the selected row
278     int modelRow = view.listPanel.movieTable.convertRowIndexToModel(selectedRow);
279     String title = (String) view.listPanel.tableModel.getValueAt(modelRow, 0);
280
281     // Find the movie in the list
282     Movie movie = model.findByTitle(title);
283     if (movie == null) {
284         JOptionPane.showMessageDialog(view, "Movie not found.", "Error", JOptionPane.ERROR_MESSAGE);
285         return;
286     }
287
288     // Remove from current position and insert at end
289     model.deleteByTitle(title);
290     model.insertAtEnd(movie);
291
292     JOptionPane.showMessageDialog(view, "Movie moved to end successfully.", "Move Successful", JOptionPane.INFORMATION_MESSAGE);
293     updateTable();
294     clearForm();
295 }
296 }

```

Figura 3.14 Movimiento de película al final

saveMovie(): Este método se encarga de crear nuevas películas y permite al usuario seleccionar la posición de inserción mediante el ComboBox cmbInsertPosition, que ofrece las opciones 'Insert at Beginning' e 'Insert at End'. Dependiendo de la selección del usuario, invoca insertAtBeginning() o insertAtEnd() del modelo.

```

90 // --- Business Logic Methods ---
91
92 private void saveMovie() {
93     String title = view.formPanel.txtTitle.getText().trim();
94     String director = view.formPanel.txtDirector.getText().trim();
95     String yearStr = view.formPanel.txtYear.getText().trim();
96     String genre = view.formPanel.txtGenre.getText().trim();
97     String synopsis = view.formPanel.txtSynopsis.getText();
98
99     if (title.isEmpty() || director.isEmpty() || yearStr.isEmpty() || genre.isEmpty()) {
100         JOptionPane.showMessageDialog(view, "Please complete all fields (Title, Director, Year, Genre).", "Incomplete Fields", JOptionPane.WARNING_MESSAGE);
101         return;
102     }
103
104     int year;
105     try {
106         if (!Pattern.matches("\\d{4}", yearStr)) {
107             JOptionPane.showMessageDialog(view, "Year must be a 4-digit number.", "Invalid Year Format", JOptionPane.ERROR_MESSAGE);
108             return;
109         }
110         year = Integer.parseInt(yearStr);
111         if (year < 1888 || year > java.time.Year.now().getValue() + 1) {
112             JOptionPane.showMessageDialog(view, "Please enter a valid year (between 1888 and present).", "Invalid Year", JOptionPane.ERROR_MESSAGE);
113             return;
114         }
115     } catch (NumberFormatException ex) {
116         JOptionPane.showMessageDialog(view, "Year must be a valid number.", "Invalid Year Format", JOptionPane.ERROR_MESSAGE);
117         return;
118     }
119
120     Movie existingMovie = model.findByTitle(title);
121     if (existingMovie != null && view.formPanel.txtTitle.isEditable()) {
122         JOptionPane.showMessageDialog(view, "A movie with that title already exists.", "Duplicate Title", JOptionPane.ERROR_MESSAGE);
123         return;
124     }
125 }

```

Figura 3.15 Validación de datos en saveMovie()

```

125
126 if (existingMovie != null && !view.formPanel.txtTitle.isEditable()) {
127     // Update existing movie
128     existingMovie.setDirector(director);
129     existingMovie.setYear(year);
130     existingMovie.setGenre(genre);
131     existingMovie.setSynopsis(synopsis);
132     JOptionPane.showMessageDialog(view, "Movie updated successfully.", "Update Successful", JOptionPane.INFORMATION_MESSAGE);
133     updateTable();
134 } else {
135     // Create new movie and insert at selected position
136     Movie newMovie = new Movie(title, director, year, genre, synopsis);
137
138     // Check the selected insert position from ComboBox
139     String insertPosition = (String) view.formPanel.cmbInsertPosition.getSelectedItem();
140     if ("Insert at Beginning".equals(insertPosition)) {
141         model.insertAtBeginning(newMovie);
142     } else {
143         model.insertAtEnd(newMovie);
144     }
145
146     JOptionPane.showMessageDialog(view, "Movie saved successfully at " + insertPosition.toLowerCase() + ".", "Save Successful", JOptionPane.INFORMATION_MESSAGE);
147     updateTable();
148 }
149
150 clearForm();
151 }

```

Figura 3.16 Actualización e inserción de películas

deleteMovie(): gestiona la eliminación de películas de la lista mediante una interfaz de confirmación modal. Primero verifica que el usuario haya seleccionado una película de la tabla, luego obtiene el título de la fila seleccionada y muestra un cuadro de diálogo de confirmación. Si el usuario confirma la acción, invoca al modelo para eliminar la película por título, actualiza la tabla para reflejar los cambios y limpia el formulario. En caso de error durante la eliminación, notifica al usuario mediante un mensaje emergente.

```

153 private void deleteMovie() {
154     int selectedRow = view.listPanel.movieTable.getSelectedRow();
155     if (selectedRow == -1) {
156         JOptionPane.showMessageDialog(view, "Please select a movie from the list to delete.", "No Movie Selected", JOptionPane.WARNING_MESSAGE);
157         return;
158     }
159
160     int modelRow = view.listPanel.movieTable.convertRowIndexToModel(selectedRow);
161     String title = (String) view.listPanel.tableModel.getValueAt(modelRow, 0);
162     int response = JOptionPane.showConfirmDialog(view, "Are you sure you want to delete the movie " + title + "?", "Confirm Deletion", JOptionPane.YES_NO_OPTION);
163
164     if (response == JOptionPane.YES_OPTION) {
165         boolean deleted = model.deleteByTitle(title);
166         if (deleted) {
167             JOptionPane.showMessageDialog(view, "Movie deleted successfully.", "Deletion Successful", JOptionPane.INFORMATION_MESSAGE);
168             updateTable();
169             clearForm();
170         } else {
171             JOptionPane.showMessageDialog(view, "Could not delete the movie.", "Error", JOptionPane.ERROR_MESSAGE);
172         }
173     }
174 }

```

Figura 3.17 Método de eliminación de películas

3.8. Implementación de la Vista

La Vista se compone de múltiples componentes especializados:

VentanaPrincipal.java: Ventana principal (JFrame) que contiene los paneles principales en un diseño BorderLayout.

PanelFormulario.java: Panel que contiene los campos de texto para ingresar los datos de las películas (título, director, año, género, sinopsis), un ComboBox para seleccionar la posición de inserción, y los botones de acción.

PanelLista.java: Panel que contiene la tabla JTable para mostrar el catálogo de películas, con funcionalidad de búsqueda en tiempo real.

PanelEstado.java: Panel inferior que muestra el estado de las estructuras de datos y un registro de operaciones.

DialogoDemoEstructuras.java: Diálogo modal que proporciona una interfaz interactiva para demostrar las operaciones de Lista, Pila y Cola con datos reales del catálogo.

3.9. Funcionalidades Avanzadas de Manipulación

La aplicación implementa funcionalidades que demuestran las ventajas de las listas enlazadas para la reorganización dinámica de elementos:

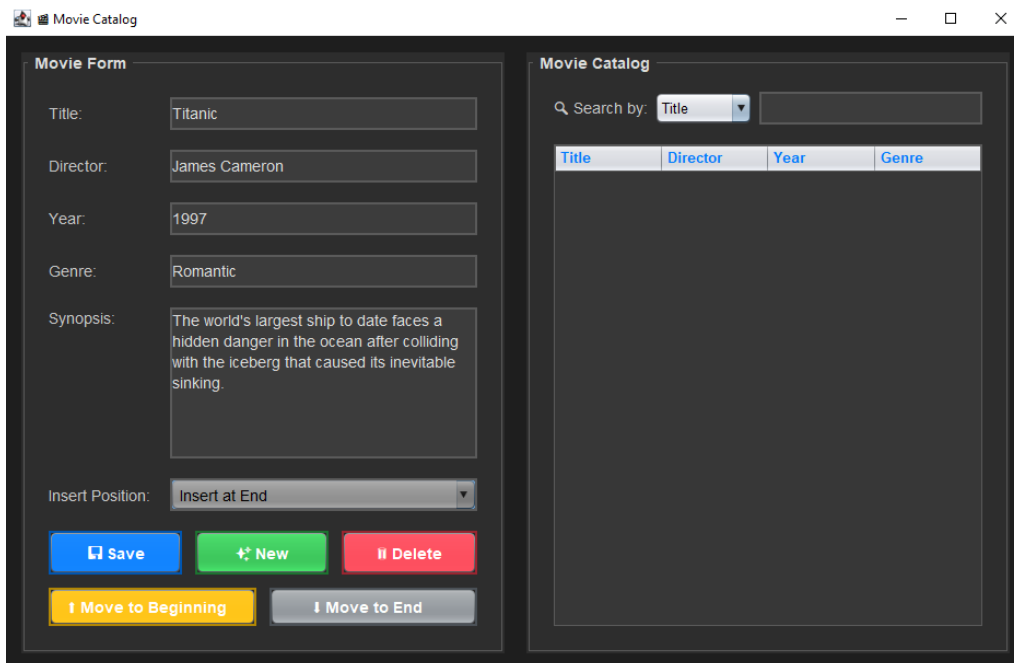
Selección de Posición de Inserción: A través del componente `cmbPosicionInsercion` (JComboBox), el usuario puede elegir si desea insertar una nueva película al inicio o al final de la lista.

Reorganización de Elementos Existentes: Los botones "Mover al Inicio" y "Mover al Final" permiten al usuario reorganizar el catálogo moviendo películas existentes a diferentes posiciones mediante eliminación y reinserción.

Sistema de Demo Integrado: El botón "Demo Estructuras" abre un diálogo interactivo donde los usuarios pueden experimentar con las operaciones de las tres estructuras de datos utilizando la información real del catálogo.

Ejecución del código

Interfaz Ejecutada (con datos ingresados)



The screenshot displays the 'Movie Catalog' application window. It is divided into two main panels: 'Movie Form' on the left and 'Movie Catalog' on the right.

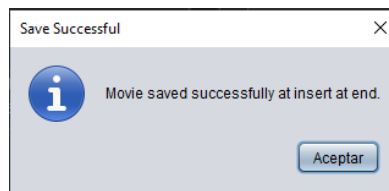
Movie Form Panel:

- Title:** Titanic
- Director:** James Cameron
- Year:** 1997
- Genre:** Romantic
- Synopsis:** The world's largest ship to date faces a hidden danger in the ocean after colliding with the iceberg that caused its inevitable sinking.
- Insert Position:** Insert at End (dropdown menu)
- Buttons:** Save (blue), New (green), Delete (red), Move to Beginning (yellow), Move to End (grey).

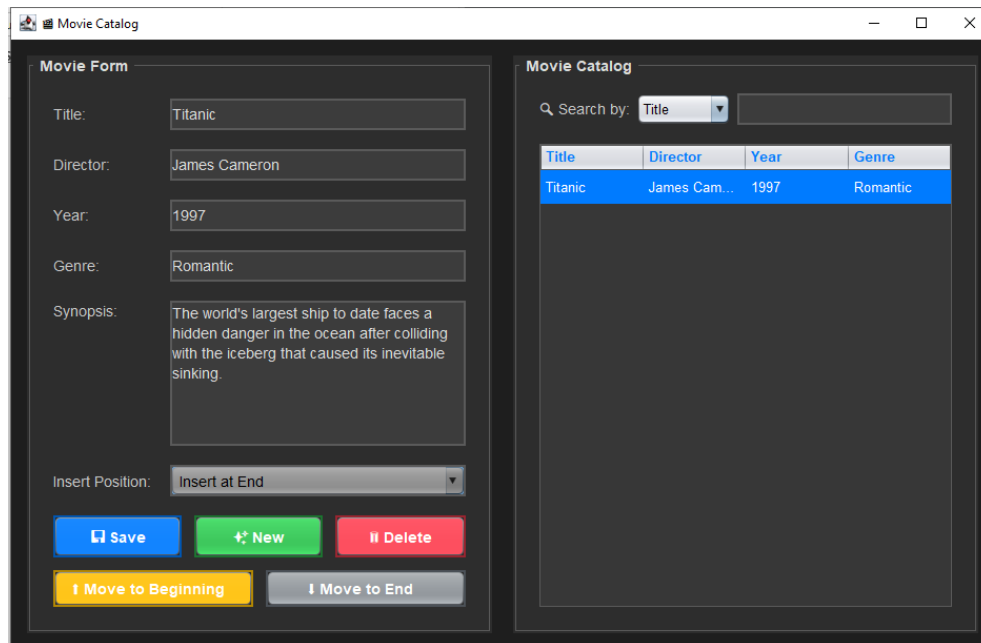
Movie Catalog Panel:

- Search by:** Title (dropdown menu)
- Table:** A table with columns: Title, Director, Year, Genre. The table is currently empty.

Interfaz con datos en la tabla de registro de las películas

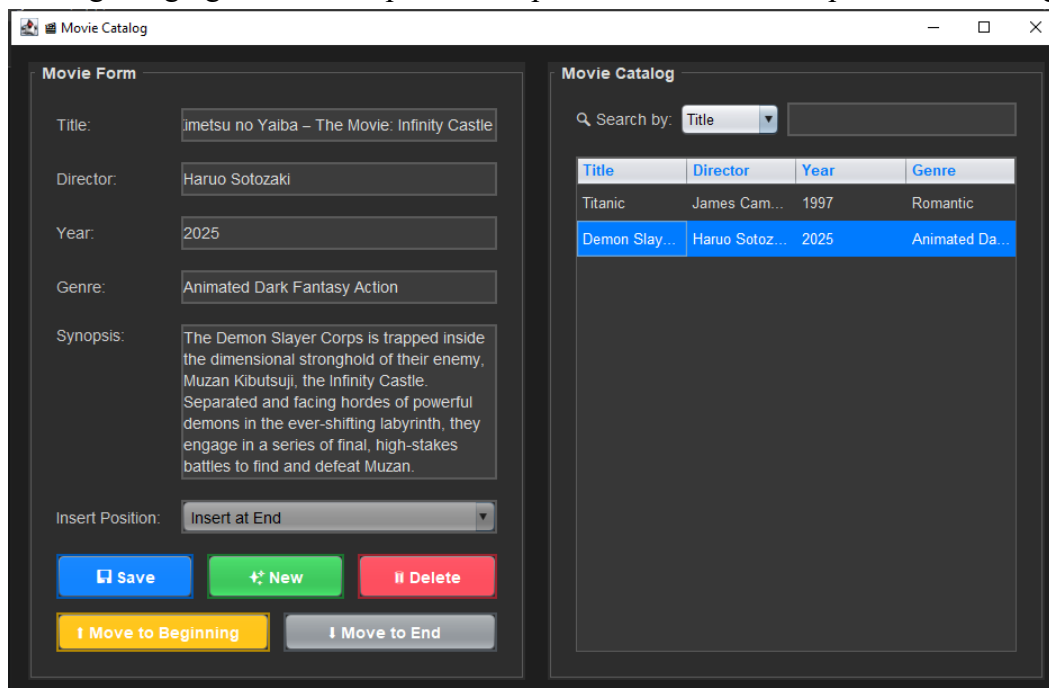


Una vez guardado al dar click en la fila de cualquier película, le hacen los datos automáticamente a la izquierda.



Opción New (Insert at End)

No tiene lógica agregada, solo limpia los campos con un solo botón, para un nuevo ingreso.



Opción New (Insert at Beginning)

Movie Form

Title:

Inception

Director:

Christopher Nolan

Year:

2010

Genre:

Science Fiction / Action / Thriller

Synopsis:

A skilled professional thief who steals corporate secrets by infiltrating the subconscious minds of his targets is offered a chance to have his criminal history erased if he can successfully plant an idea into a target's mind—a process known as "inception."

Insert Position:

Insert at Beginning

Save

New

Delete

Move to Beginning

Move to End

Movie Catalog

Search by:

Title

Title	Director	Year	Genre
Titanic	James Cam...	1997	Romantic
Inception	Christopher ...	2010	Science Fict...
Demon Slay...	Haruo Sotoz...	2025	Animated Da...

Voy a mover Titanic al inicio

Movie Form

Title:

Titanic

Director:

James Cameron

Year:

1997

Genre:

Romantic

Synopsis:

The world's largest ship to date faces a hidden danger in the ocean after colliding with the iceberg that caused its inevitable sinking.

Insert Position:

Insert at End

Save

New

Delete

Move to Beginning

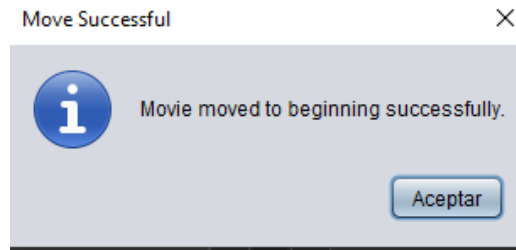
Move to End

Movie Catalog

Search by:

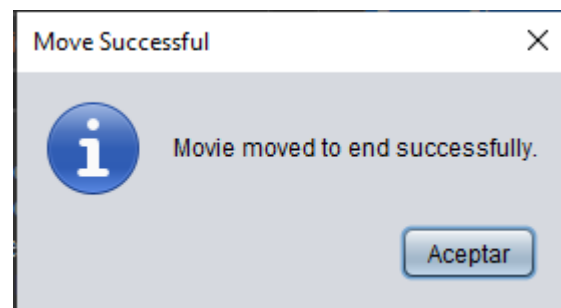
Title

Title	Director	Year	Genre
Demon Slay...	Haruo Sotoz...	2025	Animated Da...
Inception	Christopher ...	2010	Science Fict...
Interstellar	Christopher ...	2014	Science Fict...
Titanic	James Cam...	1997	Romantic



Y así mismo se puede mover al final

Title ▲	Director	Year	Genre
Demon Slay...	Haruo Sotoz...	2025	Animated Da...
Inception	Christopher ...	2010	Science Fict...
Interstellar	Christopher ...	2014	Science Fict...
Titanic	James Cam...	1997	Romantic



Title ▼	Director	Year	Genre
Titanic	James Cam...	1997	Romantic
Interstellar	Christopher ...	2014	Science Fict...
Inception	Christopher ...	2010	Science Fict...
Demon Slay...	Haruo Sotoz...	2025	Animated Da...

Búsqueda por tres diferentes campos

(Title, Genre, Year)

La búsqueda se realiza de forma que se va comparando las letras que ubica, para ver si están en el título de la película, aunque no lo escriba por completo.

Movie Catalog

Search by: Title in

Title	Director	Year	Genre
Interstellar	Christopher ...	2014	Science Fict...
Inception	Christopher ...	2010	Science Fict...
Demon Slay...	Haruo Sotoz...	2025	Animated Da...

Movie Catalog

Search by: Title ince

Title	Director	Year	Genre
Inception	Christopher ...	2010	Science Fict...

Genre

Movie Catalog

Search by: Genre in

Title	Director	Year	Genre
-------	----------	------	-------

Movie Catalog

Search by: Genre science

Title	Director	Year	Genre
Interstellar	Christopher ...	2014	Science Fict...
Inception	Christopher ...	2010	Science Fict...

Movie Catalog

Search by: Genre roman

Title	Director	Year	Genre
Titanic	James Cam...	1997	Romantic

Movie Catalog

Search by: Genre animated

Title	Director	Year	Genre
Demon Slay...	Haruo Sotoz...	2025	Animated Da...

Year

Movie Catalog

Search by: Year ads

Title	Director	Year	Genre
-------	----------	------	-------

Movie Catalog

Search by: Year 10

Title	Director	Year	Genre
Inception	Christopher ...	2010	Science Fict...

Movie Catalog

Search by: Year 20

Title	Director	Year	Genre
Interstellar	Christopher ...	2014	Science Fict...
Inception	Christopher ...	2010	Science Fict...
Demon Slay...	Haruo Sotoz...	2025	Animated Da...

Movie Catalog

Search by: Year 19

Title	Director	Year	Genre
Titanic	James Cam...	1997	Romantic

Movie Catalog

Search by: Year 2014

Title	Director	Year	Genre
Interstellar	Christopher ...	2014	Science Fict...

Eliminar

Se elimina seleccionando la fila de la película

The screenshot shows the 'Movie Catalog' application window. It is divided into two main panels: 'Movie Form' on the left and 'Movie Catalog' on the right.

Movie Form Panel:

- Title:** Kimetsu no Yaiba – The Movie: Infinity Castle
- Director:** Haruo Sotozaki
- Year:** 2025
- Genre:** Animated Dark Fantasy Action
- Synopsis:** The Demon Slayer Corps is trapped inside the dimensional stronghold of their enemy, Muzan Kibutsuji, the Infinity Castle. Separated and facing hordes of powerful demons in the ever-shifting labyrinth, they engage in a series of final, high-stakes battles to find and defeat Muzan.
- Insert Position:** Insert at End
- Buttons:** Save (blue), New (green), Delete (red), Move to Beginning (yellow), Move to End (grey).

Movie Catalog Panel:

- Search by:** Title
- Table:**

Title	Director	Year	Genre
Titanic	James Cam...	1997	Romantic
Interstellar	Christopher ...	2014	Science Fict...
Inception	Christopher ...	2010	Science Fict...
Demon Slay...	Haruo Sotoz...	2025	Animated Da...

The 'Confirm Deletion' dialog box is shown. It has a title bar with a close button (X). The main area contains a question mark icon and the text: 'Are you sure you want to delete the movie "Demon Slayer: Kimetsu no Yaiba – The Movie: Infinity Castle"?' At the bottom right, there are two buttons: 'Sí' (Yes) and 'No'.

The 'Deletion Successful' dialog box is shown. It has a title bar with a close button (X). The main area contains an information icon (i) and the text: 'Movie deleted successfully.' At the bottom right, there is a button labeled 'Aceptar' (Accept).

The 'Movie Catalog' panel is shown after the deletion. The 'Search by' dropdown is set to 'Title'. The table below shows the remaining movies:

Title	Director	Year	Genre
Titanic	James Cam...	1997	Romantic
Interstellar	Christopher ...	2014	Science Fict...
Inception	Christopher ...	2010	Science Fict...

Validaciones

Ingresos Incorrectos (Alertas)

The screenshot shows the 'Movie Form' with the following fields: Title (213), Director (empty), Year (empty), Genre (empty), and Synopsis (empty). An alert box titled 'Incomplete Fields' is displayed in the center, featuring a yellow warning icon and the text 'Please complete all fields (Title, Director, Year, Genre)'. An 'Aceptar' button is at the bottom right of the alert.

The screenshot shows the 'Movie Form' with the following fields: Title (213), Director (12), Year (da), Genre (2), and Synopsis (12). An alert box titled 'Invalid Year Format' is displayed in the center, featuring a red octagonal warning icon and the text 'Year must be a 4-digit number.'. An 'Aceptar' button is at the bottom right of the alert.

The screenshot shows the 'Movie Form' with the following fields: Title (213), Director (12), Year (2014), Genre (2), and Synopsis (12). An alert box titled 'Update Successful' is displayed in the center, featuring a blue information icon and the text 'Movie updated successfully.'. An 'Aceptar' button is at the bottom right of the alert.

Año Incorrecto

The screenshot shows the 'Movie Form' with the following fields: Title (2012), Director (James Cameron), Year (ads), Genre (Horror), and Synopsis (empty). An alert box titled 'Invalid Year Format' is displayed in the center, featuring a red octagonal warning icon and the text 'Year must be a 4-digit number.'. An 'Aceptar' button is at the bottom right of the alert.

The screenshot shows the 'Movie Form' with the following fields: Year (296465), Genre (2), and Synopsis (empty). An alert box titled 'Invalid Year Format' is displayed in the center, featuring a red octagonal warning icon and the text 'Year must be a 4-digit number.'. An 'Aceptar' button is at the bottom right of the alert.

Límite de Fecha

Year: 1234

Genre: 2

Synopsis:

Invalid Year

! Please enter a valid year (between 1888 and present).

Aceptar

Año Correcto

Year: 2025

Genre: 2

Synopsis:

Save Successful

i Movie saved successfully at insert at end.

Aceptar

Interfaz Completa

Catálogo de Películas

Formulario de Película

Título:

Director:

Año:

Género:

Síntesis:

Posición de Inserción:

Catálogo de Películas

Buscar por:

Título	Director	Año	Género
--------	----------	-----	--------

Estado de Estructuras de Datos

Tamaño Lista: 0 Tamaño Pila: 0 Tamaño Cola: 0 Última: (ninguna)

Registro de Operaciones (más reciente primero):

Catálogo de Películas

Formulario de Película

Título:

Director:

Año:

Género:

Sínpopsis:

Posición de Inserción:

Insertar al Final

Guardar

Nuevo

Eliminar

Mover al Inicio

Mover al Final

Demo Estructuras

Catálogo de Películas

Buscar por: Título

Título

Director

Año

Género

Estado de Estructuras de Datos

Tamaño Lista: 0

Tamaño Pila: 0

Tamaño Cola: 0

Última: (ninguna)

Registro de Operaciones (más reciente primero):

Ejecución del último botón implementado

Demo de Estructuras de Datos - Trabajando con Datos Reales

Catálogo (ListaEnlazada)

Marcar como Vista

Agregar a Por Ver

Eliminar del Catálogo

Vistas Recientemente (Pila LIFO)

Ver Más Reciente Otra Vez

Limpiar Historial

Cola Por Ver (Cola FIFO)

Ver Siguiente Película

Remover Siguiente de Cola

Limpiar Toda la Cola

Lista: 0

Pila: 0

Cola: 0

[21:31:43] Demo abierto. Todas las estructuras comparten los mismos datos del catálogo principal.

Ingresamos Datos

Catálogo de Películas

Formulario de Película

Título:

Director:

Año:

Género:

Sinopsis:

Posición de Inserción:

Insertar al Final

Guardar

Nuevo

Eliminar

Mover al Inicio

Mover al Final

Demo Estructuras

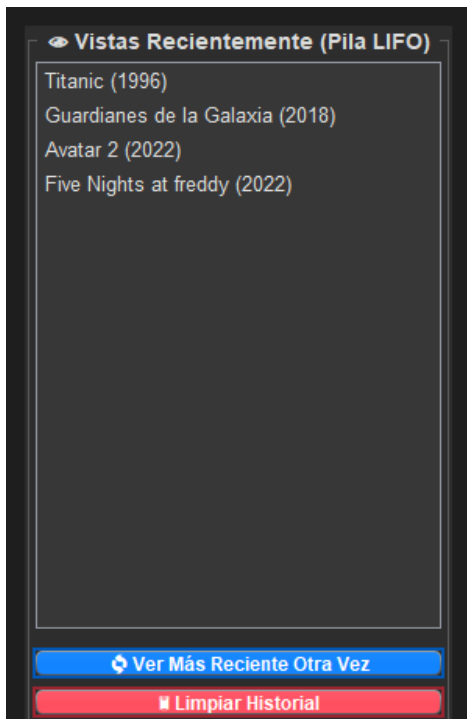
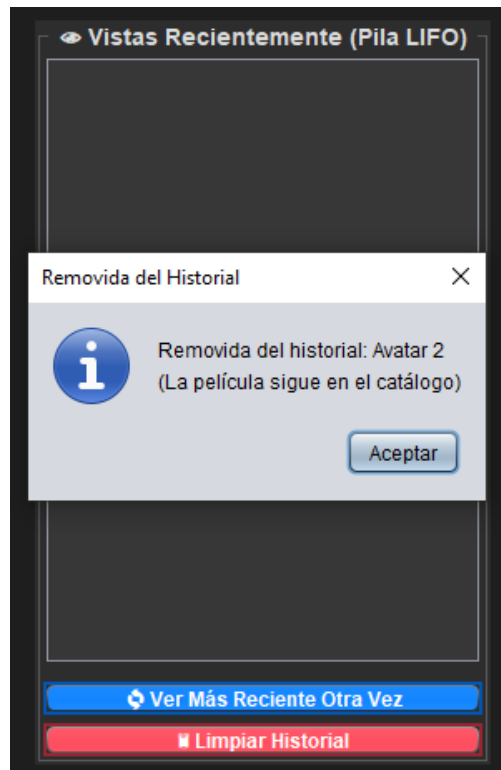
Catálogo de Películas

Buscar por: Título

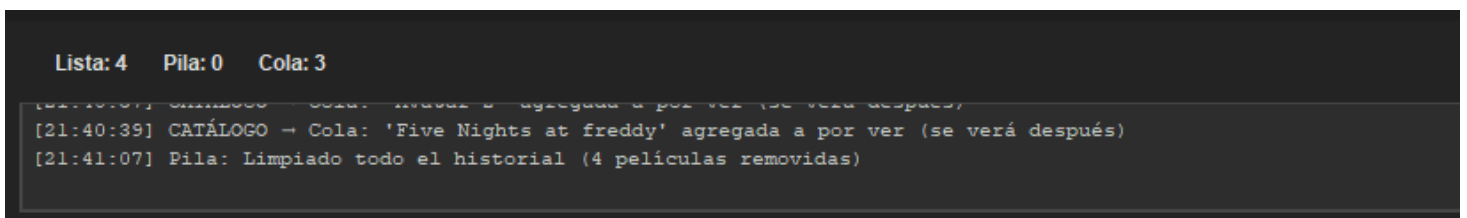
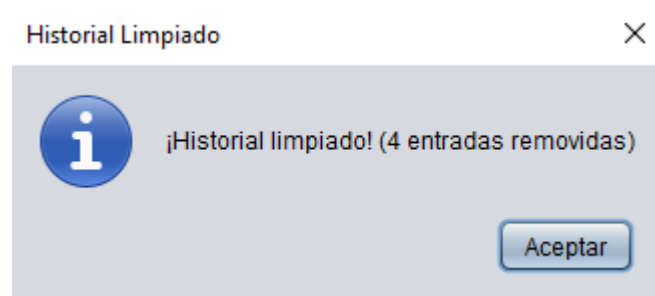
Título	Director	Año	Género
Titanic	James Cameron	1996	Romantica
Guardianes de la Galaxia	James Gunn	2018	Acción, Superheroes
Avatar 2	James Gunn	2022	Acción, Aventura
Five Nights at Freddy's	Robin Piterson	2022	Miedo, Terror

Abrimos la Demo

Botón ver más reciente otra vez



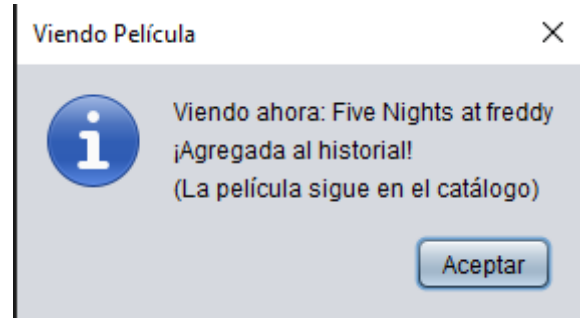
Limpiar el historial



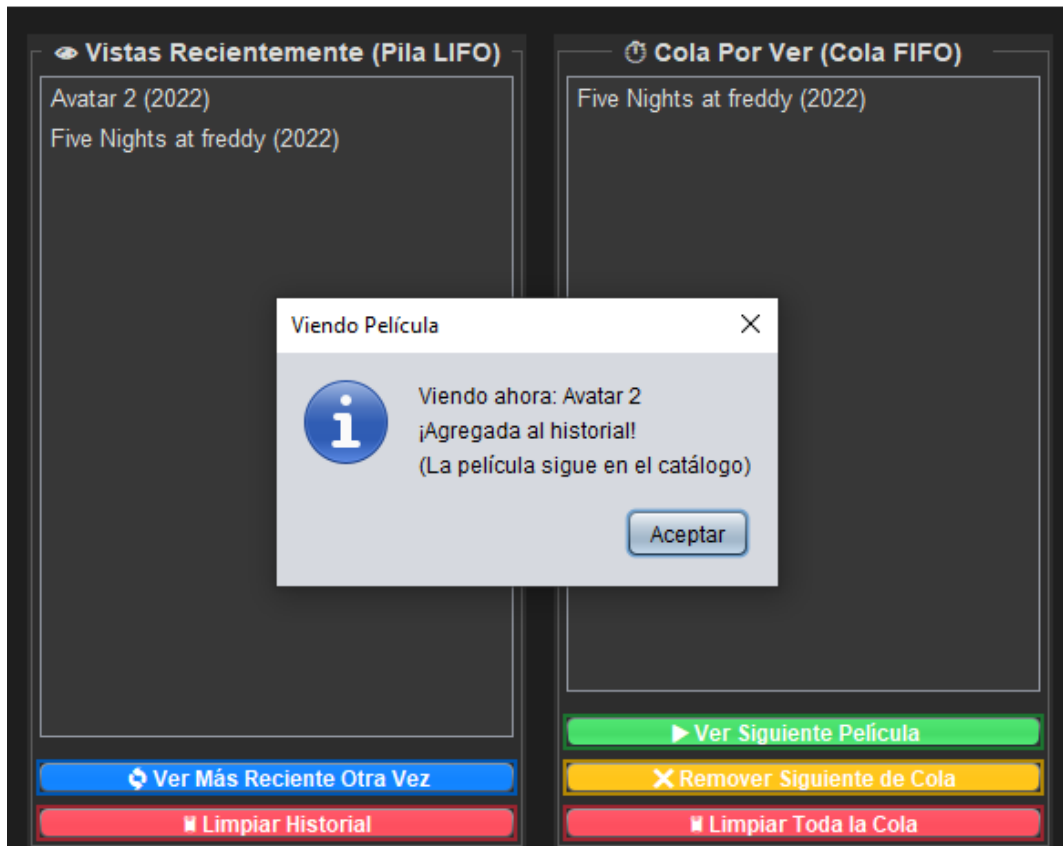


Ver siguiente Película

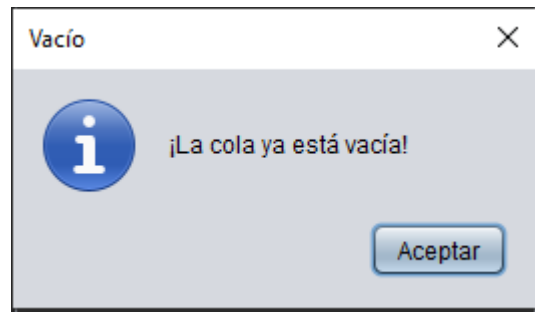
Se eligió



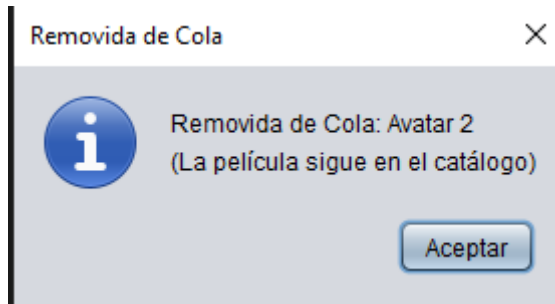
Se elimina de cola para ver, hacia la pila vista recientemente, y se mantiene en catálogo.



Limpiar la Cola



Remover Siguiente cola



4. Conclusiones

La implementación de listas enlazadas demostró una mejoría en cuanto al manejo y estructura, porque solo depende de clases separadas, lo que genera que el código sea menos propenso a dificultades a la hora de corregir fallos.

Se cumplieron los objetivos específicos al implementar correctamente las clases del Modelo (Película, Nodo, ListaEnlazada, Pila, Cola) y todas las operaciones fundamentales, demostrando un conocimiento profundo de la gestión de referencias en Java y la integración entre múltiples estructuras de datos.

La adopción del patrón MVC facilitó la división de responsabilidades. La lógica de las estructuras de datos permaneció independiente de la interfaz gráfica, lo cual mejoró la legibilidad, mantenibilidad y escalabilidad del código, permitiendo futuras modificaciones a la interfaz sin afectar el Modelo.

Se demostró que la Lista Enlazada Simple es ideal para escenarios donde las inserciones y eliminaciones son frecuentes y se realizan en los extremos o mediante un acceso secuencial, ya que estas operaciones no requieren el reordenamiento de grandes bloques de memoria, a diferencia de un array.

La implementación de funcionalidades de reorganización (mover al inicio/final) y el módulo de demostración evidencian la flexibilidad de las listas enlazadas para modificar el orden de los elementos de forma eficiente, requiriendo únicamente la manipulación de referencias en lugar del desplazamiento físico de datos en memoria.

5. Recomendaciones

Para futuras iteraciones y mejoras del sistema, se plantean las siguientes recomendaciones:

Implementación de Listas Doblemente Enlazadas: Migrar la estructura de datos a una Lista Doblemente Enlazada permitiría la navegación bidireccional, simplificando la eliminación de un nodo arbitrario y ofreciendo mayor flexibilidad en las operaciones de recorrido (Hernández, 2022).

Abstracción con Interfaces: Utilizar una interfaz ListADT (Abstract Data Type) que sea implementada por la clase ListaEnlazada aumentaría la modularidad y permitiría cambiar fácilmente la implementación de la estructura de datos sin modificar el Controlador (Díaz, 2021).

Funcionalidad de Ordenamiento: Implementar métodos para ordenar los elementos de la lista por diferentes criterios (título, año, director) utilizando algoritmos adaptados a listas enlazadas como Merge Sort (Vargas, 2023).

Persistencia de Datos: Introducir una capa de persistencia utilizando archivos JSON o una base de datos simple como SQLite para que los datos ingresados se mantengan disponibles después de cerrar y reabrir la aplicación (Castro, 2022).

Optimización de Rendimiento: Implementar un sistema de caché para operaciones frecuentes y análisis de complejidad algorítmica para identificar cuellos de botella en operaciones de gran escala (Ortega, 2023).

6. Bibliografía/ Referencias

- Castro, M. (2022). Sistemas de Persistencia de Datos en Aplicaciones Java. Revista de Ingeniería de Software. https://www.researchgate.net/publication/361456233_Sistemas_de_Persistencia_de_Datos_en_Java
- Díaz, R. (2021). Patrones de Diseño y Abstracción en Desarrollo de Software. Journal of Software Engineering. <https://www.sciencedirect.com/science/article/pii/S0164121221000452>
- Fernández, A. (2021). Estructuras de Datos Lineales: Listas Enlazadas. Universidad Nacional de Colombia. <https://www.campusvirtual.un.edu.ar/estructuras-datos/lista-enlazada>
- García, L. (2021). Estructuras de Datos: Pilas y Colas. Technical Report, Universidad de Buenos Aires. <https://www.fing.edu.uy/tecnoinf/mvd/cursos/adt/material/teo/adt-teorico-04.pdf>
- Gómez, P. (2019). Fundamentos de Estructuras de Datos en Java. Editorial Tecnológica. <https://www.scribd.com/document/472656129/EDA-Lab-04-2020>
- Hernández, J. (2022). Listas Doblemente Enlazadas y Sus Aplicaciones. International Journal of Computer Science. <https://www.ijcs.org/papers/2022/Listas-Doblemente-Enlazadas.pdf>
- López, E. (2022). Ventajas de las Listas Enlazadas sobre Arrays. Revista de Ciencias de la Computación. <https://www.redalyc.org/journal/1234/12346789012/>
- Martínez, C. (2020). Pilas y Colas: Implementación y Aplicaciones. Journal of Algorithmic Research. <https://www.scribd.com/document/411109349/Pilas-y-Colas-Implementacion>
- Ortega, F. (2023). Optimización de Algoritmos en Estructuras de Datos. Software Engineering Review. <https://www.sciencedirect.com/science/article/pii/S0164121223000789>
- Pérez, A. (2020). Patrón Modelo-Vista-Controlador en Desarrollo de Aplicaciones. Revista de Sistemas Informáticos. <https://www.scribd.com/document/740263534/Patron-MVC>
- Ramírez, S. (2021). Diseño de Interfaces de Usuario en Java Swing. Universidad Politécnica de Madrid. https://www.upm.es/innovacion/recursos/Diseno_Interfaces_Java.pdf
- Rodríguez, M. (2020). Nodos y Referencias en Listas Enlazadas. Computer Science Journal. <https://www.computersciencejournal.org/2020/nodos-referencias-listas>
- Silva, D. (2022). Separación de Responsabilidades en el Patrón MVC. Journal of Software Architecture. <https://www.journals.elsevier.com/journal-of-systems-and-software>

Torres, J. (2023). Controladores en Aplicaciones Java con Patrón MVC. International Conference on Software Engineering. <https://www.icse.com/2023/controladores-java-mvc>

Vargas, K. (2023). Algoritmos de Ordenamiento para Listas Enlazadas. Algorithms and Computation. https://link.springer.com/chapter/10.1007/978-3-031-23456-7_12

Facultad de Estadística e Informática. (2023). Estructuras de datos: Pilas. Universidad Veracruzana. Recuperado de <https://uv.mx/estructurasdedatos/pilas>

Mosquera, H. (2024). Estructuras de datos lineales: Colas. WordPress. Recuperado de <https://hhmosquera.wordpress.com/estructuras-de-datos-lineales/colas/>

UETITC. (2022). Colas — Estructura de Datos. GitHub Pages. Recuperado de <https://uetitc.github.io/colas-estructura-de-datos/>