

Práctica 2. Introducción a la API Swing de Java

Entornos de Usuario

5 de febrero de 2024

Objetivos

En esta práctica se introduce la arquitectura MVC (Modelo-Vista-Controlador) como elemento básico para el diseño y desarrollo de interfaces gráficas de usuario. También se verán algunos de los elementos básicos que se utilizan en el desarrollo de interfaces gráficas de usuarios.

Los objetivos concretos a alcanzar son:

- Descubrir la arquitectura MVC, entender sus ventajas e inconvenientes y desarrollar una aplicación sencilla utilizando esta arquitectura.
- Utilizar y practicar con algunas de las características de la API (*Application Programming Interface*) Swing de Java.

Índice

1	Introducción	2
2	El Modelo Vista Controlador (MVC)	2
2.1	El modelo de datos: la clase <code>PeliculaModel</code>	3
2.2	La vista: la clase <code>PeliculaView</code>	4
2.2.1	La clase <code>ListaPanel</code>	5
2.3	La aplicación Java	6
2.4	Ejecución de la aplicación	7
3	Ejercicio B: Respondiendo a los eventos del usuario	8
3.1	El controlador <code>PeliculaController.java</code>	8
3.2	El programa principal	9
4	Ejercicio C: Añadiendo más interactividad a la aplicación	10
4.1	La clase vista	11
4.2	La clase de control	12
5	Ejercicio D: Añadiendo un menú a nuestra aplicación	13
5.1	La clase <code>JMenuBar</code>	13
5.2	La nueva clase <code>PeliculaView</code>	14
5.3	La clase <code>PeliculaController</code> modificada	15
6	Ejercicio E	16
6.1	Entrega	17

Índice de figuras

1	Arquitectura Modelo-Vista-Controlador que aplicaremos a nuestra aplicación	3
2	Estructura del proyecto <code>Aplicacion1</code> en NetBeans	4
3	Salida de consola producida por la aplicación <code>Aplicacion1</code>	8
4	Salida de consola producida por la <code>Aplicacion5</code>	17
5	Salida de consola producida por la <code>Aplicacion5</code>	18

Índice de listados

1	Código de la vista <code>PeliculaView.java</code>	4
2	Código de la vista <code>ListaPanel.java</code>	6
3	Código de la aplicación <code>Aplicacion1.java</code>	7
4	La clase controlador <code>PeliculaController.java</code> con gestión de eventos de ventana. .	9
5	Código de la aplicación “3” <code>JavaApplication.java</code>	9
6	La nueva vista <code>PeliculaView.java</code>	11
7	La clase controlador <code>PeliculaController.java</code> con gestión de eventos de ventana y del botón “Salir”.	12
8	La clase <code>PeliculaMenu.java</code> que implementa la barra de menú de nuestra aplicación. .	14
9	La clase <code>PeliculaView.java</code> ampliada con el menú de la aplicación.	14
10	La clase <code>PeliculaController.java</code> modificada para responder a los eventos del menú. .	15

1 Introducción

La primera parte de esta práctica, al igual que la práctica 1, está formada por una serie de ejercicios dirigidos orientados a introducir algunos de los elementos básicos que proporciona la API Swing de Java para el desarrollo de interfaces gráficas de usuario.

Como en la práctica 1, la forma correcta de abordar la realización de estos ejercicios es:

- Lea atentamente en qué consiste el ejercicio, qué se espera de su realización y cualquier otra explicación relevante.
- Siga paso a paso las indicaciones del ejercicio, asegurándose en cada paso de que entiende qué es lo que se está haciendo y por qué.
- No aborde un paso posterior sin haber completado con éxito los pasos anteriores. Si encuentra problemas para entender o llevar a cabo algún punto, recurra a su Profesor.
- Cuando tenga que introducir código, no utilice la función *Cut & Paste*. No va a ahorrar tiempo ni esfuerzo y además perderá la oportunidad de adquirir experiencia en el lenguaje Java y en el uso del editor inteligente que integra NetBeans.
- La complejidad de los ejercicios aumenta a lo largo de la sesión. No cambie el orden de realización de los ejercicios ni inicie un ejercicio sin haber completado satisfactoriamente el ejercicio anterior. Esta norma es importante, ya que en cada ejercicio se da por supuesto que no es necesario explicar con detalle determinados pasos u operaciones que se han realizado en ejercicios anteriores.

En esta práctica utilizaremos como modelo las clases `Pelicula` y `Coleccion` (`PeliculaModel`) de la primera práctica ligeramente modificadas y emplearemos la arquitectura MVC y los elementos, principalmente, de la API Swing para construir varias interfaces gráficas de usuario de complejidad creciente. También haremos uso de algunos elementos de la API AWT de Java para completar nuestra aplicación.

Nota: Para consultar los métodos, atributos y clases de la API `javax.swing` que vamos a utilizar a lo largo de esta práctica puedes acceder a: <http://docs.oracle.com/javase/7/docs/api/index.html>.

2 El Modelo Vista Controlador (MVC)

Esta arquitectura software es muy útil para el diseño de programas interactivos con interfaz gráfica. La principal característica de este, es la separación de la lógica de negocio de la interfaz de usuario. Esto proporciona dos características muy importantes:

- facilita la evolución por separar de ambos aspectos e
- incrementa la reutilización y flexibilidad de los programas.

El flujo de control del MVC es:

1. El usuario realiza una acción en la interfaz
2. El controlador trata el evento de entrada (Previamente se ha registrado)
3. El controlador notifica al modelo la acción del usuario, lo que puede implicar un cambio del estado del modelo (si no es una mera consulta)
4. Se genera una nueva vista. La vista toma los datos del modelo
5. El modelo no tiene conocimiento directo de la vista
6. La interfaz de usuario espera otra interacción del usuario, que comenzará otro nuevo ciclo

¿Cómo vamos a aplicar el modelo vista controlador (MVC) a nuestra aplicación? Para aplicar el MVC es recomendable utilizar tres clases tal como se muestra en el diagrama siguiente:

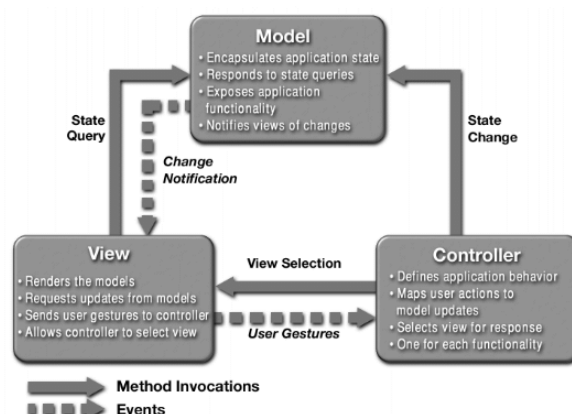


Figura 1: Arquitectura Modelo-Vista-Controlador que aplicaremos a nuestra aplicación

- La clase `PeliculaView` describe la interfaz de usuario: la pantalla de visualización y el menú. Cada uno de estos elementos se implementará en su propia clase.
- La clase `PeliculaModel`, es la que modela el comportamiento del objeto real. En nuestro caso el modelo sólo se encarga de mantener y mostrar una lista de películas.
- La clase `PeliculaController`, que será la intermediaria entre la vista y el modelo. Se encargará de tratar todo los eventos que se produzcan en la vista (responder a los *clicks* sobre los botones, **Salir** de un menú, etc). El controlador no aporta funcionalidad, sólo aporta el código necesario para que el modelo y la vista se comuniquen.

Con objeto de simplificar el desarrollo de la práctica, junto con el enunciado se suministra el fichero `Aplicacion1.zip` con el esqueleto básico del proyecto. Descargue el fichero y descomprímalo dentro de la carpeta en la que NetBeans almacena sus proyectos (generalmente `NetBeansProjects` en el directorio personal del usuario). Después acceda al proyecto desde NetBeans con **File** → **Open Project...**. En la figura 2 se muestra la estructura completa de paquetes y ficheros del proyecto que describiremos con detalle en los siguientes apartados.

2.1 El modelo de datos: la clase `PeliculaModel`

Las clases `PeliculaModel` y `Pelicula` se proporcionan junto con el proyecto que has desempaquetado en el punto anterior. Se trata de las mismas clases de la práctica 1:

- `Coleccion`, renombrada como `PeliculaModel` para destacar el hecho de que es el “modelo” de nuestra aplicación.
- La clase `Pelicula`.

Es importante notar que esta clase se reutilizará sin modificaciones en lo que queda de práctica. Esto es lógico, ya que el modelo de datos es independiente de la forma en que se visualicen los datos que contiene. Esta es la situación deseable y en general la más frecuente.

Por tanto, para “mejorar” nuestra aplicación, sólo será necesario crear las clases que se encargan de la visualización de la información (`PeliculaView`) y de gestionar los eventos producidos por la interacción con el usuario (`PeliculaController`) que veremos más adelante.

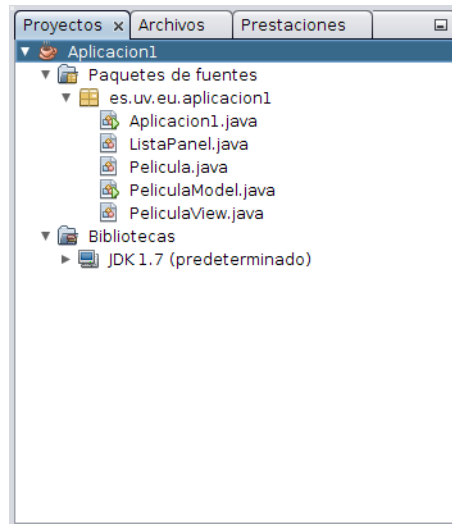


Figura 2: Estructura del proyecto Aplicacion1 en NetBeans

2.2 La vista: la clase PeliculaView

La clase vista se encarga de todos los aspectos relativos a la visualización de los datos. Aunque en este caso se trata de una clase muy simple que se limita a mostrar los datos en una ventana, ya tiene algunas de las características que son comunes a este tipo de clases:

- Cuentan con un atributo que es una referencia al modelo, en este caso `PeliculaModel`, para acceder a los datos que debe visualizar.
- Esta referencia al modelo se inicializa a través del constructor de la clase.
- Deriva (`extends`) de la clase `JFrame`, por lo que es y se comporta como una “ventana” sobre la que mostraremos todos los elementos visuales.
- Instancia un objeto de tipo *layout* (en este caso `BorderLayout`) y mediante el método `setLayout` lo establece como organizador de los elementos (componentes o contenedores) de la ventana.
- Instancia un objeto de la clase `ListaPanel` que deriva de la clase `JPanel` que detallaremos en el apartado 2.2.1. Este objeto es añadido al *frame* mediante el método `add`. Es importante destacar que el uso combinado de paneles y objetos del tipo *layout*, es el mecanismo básico por el cual se organizan los elementos visuales en la ventana.

Nota:

1. ¿A qué clase pertenece el método `add`? Busca en la documentación de la API Swing.
2. ¿Qué sucedería si comentamos la línea 25 del listado 1? ¿Obtendríamos la misma disposición de los componentes? ¿Por qué?

- Declara el método `public void setActionListener(ActionListener actionListener)`, que de momento no hace nada pero que usaremos en el futuro para asignar a cada elemento de la vista el controlador que debe responder a los eventos del usuario.

En el listado 1 se muestra el código Java de esta clase.

Listado 1: Código de la vista `PeliculaView.java`.

```
1 package es.uv.eu.aplicacion1;
2
3 import java.awt.BorderLayout;
4 import java.awt.Font;
5 import java.awt.event.ActionListener;
6 import javax.swing.JFrame;
7 import javax.swing.JLabel;
8
9 /**
```

```

10  *
11  * @author Entornos de usuario
12  */
13
14  public class PeliculaView extends JFrame {
15
16      private PeliculaModel peliculas;
17      private ListaPanel listaPanel;
18      private PeliculaMenu peliculaMenu;
19      private JButton exit;
20
21      public PeliculaView(PeliculaModel peliculas) {
22          // Invoca al constructor de la superclase Frame
23          super("Aplicacion 1: Lista de peliculas");
24
25          this.peliculas = peliculas;
26          this.setLayout(new BorderLayout());
27
28          // Tamanyo del Frame
29          this.setSize(800,400);
30
31          //Boton
32
33          exit= new JButton("Salir");
34
35          // Cabecera identificativa de la aplicacion
36          JLabel label = new JLabel("Lista de películas:");
37          label.setFont(new Font("Sans", Font.BOLD, 14));
38
39          // El panel con la lista de peliculas
40          listaPanel = new ListaPanel(peliculas);
41          listaPanel.mostrar();
42
43          // Anyade todos los componentes al frame
44          this.add(label, BorderLayout.NORTH);
45          this.add(listaPanel, BorderLayout.CENTER);
46          this.add(exit, BorderLayout.SOUTH);
47      }
48
49      public void setActionListener(ActionListener actionListener) {}
50  }

```

2.2.1 La clase ListaPanel

La clase `ListaPanel` se muestra en el listado 2. Es interesante señalar varias características de esta clase:

- Deriva de la clase `JPanel`. Esta clase genera objetos contenedor que no son visibles y cuya finalidad es albergar componentes visibles.
- Al constructor se le pasa como argumento el modelo.
- Contiene un componente visible de la clase `JTextArea` que se utilizará para visualizar la lista de películas del modelo de datos. Para añadir una barra de scroll al área de texto, utilizamos un contenedor de la clase `JScrollPane`, donde al constructor de dicha clase le pasamos el objeto de tipo `JTextArea` como argumento. Finalmente este contenedor se añade a el contenedor `JPanel` mediante el método `add.add`.

Nota: Contesta a las siguientes cuestiones:

1. ¿A qué clase pertenece el método `add`? Busca en la documentación de la API Swing.
2. ¿Puede un contenedor de tipo `JPanel` contener a otros contenedores del mismo tipo?

- La clase `ListaPanel` también define un método `mostrar` que se utiliza para introducir en el componente `JTextArea` la descripción de cada película. Para ello, el método utiliza el método `toString()` `PeliculaModel` para obtener una representación del objeto. Es importante hacer notar que se ha añadido un `BorderLayout` como gestor de contenidos.

Nota: Contesta a las siguientes cuestiones:

1. ¿Qué pasaría si hubiésemos utilizado un `FlowLayout` en vez de un `BorderLayout`?
2. ¿Cuál es la ventaja de utilizar un `BorderLayout` frente a un `FlowLayout`?

Listado 2: Código de la vista `ListaPanel.java`.

```

1 package es.uv.eu.aplicacion1;
2 import java.awt.Color;
3 import java.awt.Font;
4 import javax.swing.JPanel;
5 import javax.swing.JScrollPane;
6 import javax.swing.JTextArea;
7 /**
8  *
9  * @author Entornos de Usuario
10 */
11 public class ListaPanel extends JPanel {
12     private PeliculaModel peliculas;
13     private JTextArea textArea;
14     private JScrollPane scroll;
15
16     public ListaPanel(PeliculaModel peliculas) {
17
18         this.peliculas = peliculas;
19
20         setLayout(new BorderLayout());
21         textArea = new JTextArea(20,60);
22         textArea.setEditable(false);
23         textArea.setBackground(Color.lightGray);
24         textArea.setForeground(Color.black);
25         textArea.setFont(new Font("Sans", Font.BOLD, 12));
26         scroll = new JScrollPane(textArea);
27
28         this.add(scroll, BorderLayout.CENTER)
29         this.setVisible(true);
30     }
31
32     public void mostrar() {
33         textArea.append(peliculas.toString());
34     }
35
36 }
```

2.3 La aplicación Java

El método `main` de la aplicación se implementa en la clase `Aplicacion1` y tiene una estructura muy simple:

- Primero instancia un objeto de la clase `PeliculaModel` y añade al objeto la lista de películas invocando al método `creaPelículas` definido en la propia clase.
- Después instancia un objeto de la clase `PeliculaView`, al que se le pasa el modelo a través del constructor.

El listado 3 muestra el código completo de esta clase.

Listado 3: Código de la aplicación `Aplicacion1.java`.

```

1 package es.uv.eu.aplicacion1;
2 /**
3  * JavaApplication: controlador del programa
4  *
5  * @author Entornos de Usuario
6  */
7 public class Aplicacion1 {
8     /**
9     *
10    * @param args the command line arguments
11    */
12    public static void main(String[] args) {
13
14        PeliculaModel model = new PeliculaModel();
15        creaPelículas(model);
16
17        PeliculaView view = new PeliculaView(model);
18
19        view.setVisible(true);
20    }
21
22    static void creaPelículas(PeliculaModel peliculas) {
23        peliculas.add(new Pelicula("2001: Una Odisea en el Espacio",
24            "Stanley Kubrick", 1968, Pelicula.CIENCIA_FICCION));
25        peliculas.add(new Pelicula("2046", "Wong Kar Wai", 2004,
26            Pelicula.CIENCIA_FICCION));
27        peliculas.add(new Pelicula("Aeon Flux", "Karyn Kusama", 2005,
28            Pelicula.CIENCIA_FICCION));
29        peliculas.add(new Pelicula("Alien, el octavo pasajero",
30            "Ridley Scott", 1979, Pelicula.CIENCIA_FICCION));
31        peliculas.add(new Pelicula("Blade Runner", "Ridley Scott", 1982,
32            Pelicula.CIENCIA_FICCION));
33        peliculas.add(new Pelicula("Contact", "Robert Zemeckis", 1997,
34            Pelicula.CIENCIA_FICCION));
35        peliculas.add(new Pelicula("Las crónicas de Riddick",
36            "David Twohy", 2004, Pelicula.CIENCIA_FICCION));
37        peliculas.add(new Pelicula("Dune", "David Lynch", 1984,
38            Pelicula.CIENCIA_FICCION));
39        peliculas.add(new Pelicula("La guerra de los mundos",
40            "Byron Haskin", 1953, Pelicula.CIENCIA_FICCION));
41        peliculas.add(new Pelicula("El quinto elemento",
42            "Jean Luc Besson", 1997, Pelicula.CIENCIA_FICCION));
43    }
44 }

```

2.4 Ejecución de la aplicación


Ejecuta el programa. Para ello abre el menú contextual mediante un click con el botón derecho sobre la clase `Aplicacion1` y seleccione **Ejecutar archivo**. Observa el resultado, que debe ser similar al mostrado en la figura 3.

Nota: Puede que te resulte curioso el hecho de que las películas no se muestren en el mismo orden en el que fueron introducidas en el constructor de la clase `PeliculaModel`. ¿Por qué? Intenta buscar una explicación a este comportamiento aparentemente extraño.




Figura 3: Salida de consola producida por la aplicación `Aplicacion1`.

3 Ejercicio B: Respondiendo a los eventos del usuario

En estos momentos nuestra aplicación no es muy sensible a los deseos del usuario. Por ejemplo, no responde a los controles para cerrar la ventana () , por lo que el objetivo de este ejercicio es introducir en la aplicación un controlador que permita responder a esta acción del usuario.

Sigue los siguiente pasos:

1. Copia el proyecto `Aplicacion1` con el nombre `Aplicacion2`. Para ello selecciona el proyecto `Aplicacion1` con un click del botón derecho y en el menú contextual selecciona .
2. Crea en el nuevo proyecto la clase vacía `PeliculaController` que completaremos a continuación.

3.1 El controlador `PeliculaController.java`

En el listado 4 se proporciona el código de la clase `PeliculaController` y cuyas características principales son:

- Cuenta con dos atributos privados `PeliculaModel` y `PeliculaView`. Estos atributos se inicializan a través del constructor de la clase y permitirán que el controlador interactúe tanto con el modelo como con la vista. Esta es una de las características de la arquitectura M-V-C.
- Define una **clase empotrada**, `PeliculaControllerWindowListener` que deriva de la clase `WindowAdapter`. Esta clase es la que sobrescribe el método `windowClosing` de `WindowAdapter` y que es el encargado de responder a evento de cierre de ventana generado por el usuario. En este caso se limita a finalizar la ejecución de la aplicación (mediante la llamada a `System.exit()`) devolviendo el control al sistema operativo anfitrión.

Nota: Una **clase empotrada** se define dentro del ámbito (*scope*) de otra clase y por tanto no se implementa dentro de su propio fichero `NombreClase.java`. Como consecuencia, sólo se puede instanciar dentro del ámbito de la clase que la contiene.

Nota: No es necesario que el *listener* se defina como una clase empotrada dentro del controlador. En este caso se ha optado en el diseño por esta estrategia de implementación. En la literatura es posible encontrar muchas otras formas de abordar este diseño y que van desde utilizar clases “normales” (que implemente interfaces) hasta hacer que el propio controlador derive de la clase *WindowAdapter*.

Nota: Para implementar el *listener* hemos hecho que esta clase derive de *WindowAdapter*, sin embargo se podría haber optado por otra segunda opción, donde el *listener* implementase una interfaz *WindowListener* ¿Por qué se ha optado por la primera opción? ¿Cuáles son las ventajas?

- Asocia o registra una instancia de la clase *PeliculaControllerWindowListener* (el oyente o listener) con la vista mediante el método *addWindowListener* (heredado de la clase *JFrame*) para que nuestra vista responda a ese evento de usuario.

Listado 4: La clase controlador *PeliculaController.java* con gestión de eventos de ventana.

```

1 package es.uv.eu.aplicacion2;
2
3 import java.awt.event.WindowAdapter;
4 import java.awt.event.WindowEvent;
5
6 /**
7  *
8  * @author Entornos de usuario
9  */
10 public class PeliculaController {
11     private PeliculaModel model;
12     private PeliculaView view;
13
14     public PeliculaController (PeliculaModel model, PeliculaView view) {
15         this.model = model;
16         this.view = view;
17
18         view.addWindowListener(new PeliculaControllerWindowListener());
19     }
20
21     // Clases empotradas
22     class PeliculaControllerWindowListener extends WindowAdapter {
23         @Override
24         public void windowClosing(WindowEvent e) {
25             System.out.println("PeliculaController : Cerrar ventana.");
26             System.exit(0);
27         }
28     }
29 }

```

3.2 El programa principal

El programa principal debe ahora crear una instancia de la clase *PeliculaController*. En el listado 5 se muestra el código modificado (línea 18) de la clase principal. Recuerda la estructura de esta clase, ya que de ahora en adelante constituirá la estructura típica de nuestras aplicaciones.

Listado 5: Código de la aplicación “3” *JavaApplication.java*.

```

1 package es.uv.eu.aplicacion2;
2 /**
3  * JavaApplication: controlador del programa
4  */

```

```

5  * @author Entornos de Usuario
6  */
7  public class Aplicacion2 {
8      /**
9       *
10      * @param args the command line arguments
11      */
12      public static void main(String[] args) {
13
14          PeliculaModel model = new PeliculaModel();
15          creaPelículas(model);
16
17          PeliculaView view = new PeliculaView(model);
18          PeliculaController controller = new PeliculaController(model, view);
19
20          view.setVisible(true);
21      }
22
23      static void creaPelículas(PeliculaModel peliculas) {
24          peliculas.add(new Pelicula("2001: Una Odisea en el Espacio",
25              "Stanley Kubrick", 1968, Pelicula.CIENCIA_FICCION));
26          peliculas.add(new Pelicula("2046", "Wong Kar Wai", 2004,
27              Pelicula.CIENCIA_FICCION));
28          peliculas.add(new Pelicula("Aeon Flux", "Karyn Kusama", 2005,
29              Pelicula.CIENCIA_FICCION));
30          peliculas.add(new Pelicula("Alien, el octavo pasajero",
31              "Ridley Scott", 1979, Pelicula.CIENCIA_FICCION));
32          peliculas.add(new Pelicula("Blade Runner", "Ridley Scott",
33              1982, Pelicula.CIENCIA_FICCION));
34          peliculas.add(new Pelicula("Contact", "Robert Zemeckis", 1997,
35              Pelicula.CIENCIA_FICCION));
36          peliculas.add(new Pelicula("Las crónicas de Riddick",
37              "David Twohy", 2004, Pelicula.CIENCIA_FICCION));
38          peliculas.add(new Pelicula("Dune", "David Lynch", 1984,
39              Pelicula.CIENCIA_FICCION));
40          peliculas.add(new Pelicula("La guerra de los mundos",
41              "Byron Haskin", 1953, Pelicula.CIENCIA_FICCION));
42          peliculas.add(new Pelicula("El quinto elemento",
43              "Jean Luc Besson", 1997, Pelicula.CIENCIA_FICCION));
44      }
45  }

```


4 Ejercicio C: Añadiendo más interactividad a la aplicación

Completaremos esta práctica inicial introduciendo un elemento muy frecuente en las interfaces gráficas de usuario: un botón con el que el usuario puede definir una acción dentro de la aplicación. En este caso añadiremos un botón **Salir** bajo el panel `ListaPanel` que muestra los datos y modificaremos el controlador para que incorpore un *listener* que responda al evento.

Las modificaciones en nuestro código deben apuntar en dos direcciones:

- Puesto que estamos “mejorando” el aspecto visual de nuestra aplicación, será necesario ampliar la clase `PeliculaView` para que incorpore los elementos adicionales (en este caso un botón).
- Por otra parte, ya que aumentamos la interactividad con el usuario, será necesario añadir los elementos (*listeners*) que respondan a los nuevos eventos. En este caso debemos añadir un *listener* para responder al evento *Click sobre el botón salir*.

Para realizar este ejercicio sigue los siguientes pasos:

1. Copia el proyecto `Aplicacion2` con el nombre `Aplicacion3`. Para ello selecciona el proyecto `Aplicacion2` con un click del botón derecho y en el menú contextual selecciona  .

4.1 La clase vista

En el listado 6 se muestra el código de la nueva clase `PeliculaView`. Podemos notar las siguientes diferencias respecto a la versión anterior de esta clase:

- Se crea un nuevo objeto `exit` del tipo `JButton` .
- Se asocia la acción `buttonExit` al botón. Esta acción es una simple etiqueta que se pasará a través del evento correspondiente y que nos permitirá que el *listener* sepa qué botón se presionó.
- Se añade (*add*) el botón al marco. En ausencia de otro tipo de indicaciones (que veremos con detalle a lo largo del curso), el nuevo botón se visualizará bajo el `JTextArea` .
- Se añade funcionalidad al método `setActionListener` para que asocie un *listener* al botón `Exit` mediante la llamada al método `addActionListener` de la clase `JButton` .

Listado 6: La nueva vista `PeliculaView.java`.

```

1
2 package es.uv.eu.aplicacion3;
3
4 import java.awt.BorderLayout;
5 import java.awt.Font;
6 import java.awt.event.ActionListener;
7 import javax.swing.JButton;
8 import javax.swing.JFrame;
9 import javax.swing.JLabel;
10
11 *
12
13 @author Entornos de Usuario
14 /
15 public class PeliculaView extends JFrame {
16
17     private PeliculaModel peliculas;
18     private ListaPanel listaPanel;
19     private JButton exit;
20     private PeliculaMenu peliculaMenu;
21
22     public PeliculaView(PeliculaModel peliculas) {
23         // Invoca al constructor de la superclase Frame
24         super("Aplicacion 1: Lista de peliculas");
25
26         this.peliculas = peliculas;
27         this.setLayout(new BorderLayout());
28
29         // Tamano del Frame
30         this.setSize(700,400);
31
32         // Cabecera identificativa de la aplicacion
33         JLabel label = new JLabel("Lista de ípeliculas:");
34         label.setFont(new Font("Sans", Font.BOLD, 14));
35
36         // El panel con la lista de ípeliculas
37         listaPanel = new ListaPanel(peliculas);
38         listaPanel.mostrar();
39
40         // El boton de salida
41         exit = new JButton("Salir");
42         exit.setActionCommand("buttonExit");
43
44         // Anyade todos los componentes al frame
45         this.add(label, BorderLayout.NORTH);
46         this.add(listaPanel, BorderLayout.CENTER);

```

```

47     this.add(exit, BorderLayout.SOUTH);
48
49
50     this.setVisible(true);
51 }
52 public void setActionListener(ActionListener actionListener) {
53     exit.addActionListener(actionListener);
54 }
55 }

```

4.2 La clase de control

Para responder al evento del nuevo botón, debemos ampliar la clase `PeliculaController` con un `ActionListener`. En el listado 7 se muestra la nueva clase `PeliculaController`. Para responder al click sobre el botón **Salir**, hemos añadido la clase empotrada `PeliculaControllerActionListener` que implementará la interface `ActionListener`. El `getter` `ActionEvent.getActionCommand()` se utiliza para obtener la etiqueta que asociamos al botón **Salir**.

Una vez completado el código, pruebe la aplicación y cómo responde a todos los eventos previstos.

Nota: Una interfaz en Java es una colección de métodos abstractos y propiedades. En ellas se especifica qué se debe hacer pero no su implementación. Serán las clases que implementen estas interfaces las que describan la logica del comportamiento de los métodos.

El uso de interfaces proporciona las siguientes ventajas:

- Organiza la programación.
- Obliga a que ciertas clases utilicen los mismos métodos (nombres y parámetros).
- Establece relaciones entre clases que no estén relacionadas.

Listado 7: La clase controlador `PeliculaController.java` con gestión de eventos de ventana y del botón "Salir".

```

1  package es.uv.eu.aplicacion3;
2
3  import java.awt.event.ActionEvent;
4  import java.awt.event.ActionListener;
5  import java.awt.event.WindowAdapter;
6  import java.awt.event.WindowEvent;
7  /**
8   *
9   * @author Entornos de usuario
10  */
11  public class PeliculaController {
12      private PeliculaModel model;
13      private PeliculaView view;
14
15      public PeliculaController (PeliculaModel model, PeliculaView view) {
16          this.model = model;
17          this.view = view;
18
19          view.addWindowListener(new PeliculaControllerWindowListener());
20          view.addActionListener(new PeliculaControllerActionListener ());
21      }
22
23      // Clases empotradas
24      class PeliculaControllerWindowListener extends WindowAdapter {
25          @Override
26          public void windowClosing(WindowEvent e) {
27              System.out.println (" PeliculaController : Cerrar ventana.");
28              System.exit(0);

```

```

29     }
30 }
31 class PeliculaControllerActionListener implements ActionListener {
32     @Override
33     public void actionPerformed(ActionEvent ae) {
34         String command = ae.getActionCommand();
35         switch (command) {
36             case "buttonExit":
37                 System.out.println(" PeliculaController : Boton 'Salir' . ");
38                 System.exit(0);
39                 break;
40             default :
41                 System.out.println(" PeliculaController : Comando '" + command + "' no reconocido.");
42                 break;
43         }
44     }
45 }
46 }

```

5 Ejercicio D: Añadiendo un menú a nuestra aplicación

Nuestra aplicación ya responde a muchas de las acciones del usuario que nos resultan familiares. Sin embargo, carece de un elemento esencial en las aplicaciones gráficas de usuario actuales: un menú. En este ejercicio vamos a añadir una barra de menú a nuestra aplicación con una única entrada de menú `Fichero` que a su vez tiene un solo item `Salir`. Para llevar a cabo esta tarea, será necesario realizar una serie de cambios en nuestra aplicación:

- Para implementar la barra de menú, crearemos una nueva clase `PeliculaMenu` que extiende directamente de la clase `JMenuBar`.
- Será necesario modificar la clase `PeliculaView` para que incorpore el nuevo elemento en la vista y sea capaz de asignarle un controlador modificando el método `setActionListener`.
- Modificaremos la clase `PeliculaController` para que responda a la acción del usuario en el menú.

Como en los ejemplos anteriores, copia el proyecto `Aplicacion3` con el nombre `Aplicacion4`. Para ello selecciona el proyecto `Aplicacion3` con un click del botón derecho y en el menú contextual selecciona

Copiar...

5.1 La clase `JMenuBar`

En el listado 8 se muestra el listado de la clase `PeliculaMenu` que implementa la barra de menú de nuestra aplicación. Fíjate que esta clase hereda (extends) de la clase `JMenuBar`.

Nota: En Java, el menú de una aplicación se crea a través de una barra de menú (clase `JMenuBar`) a la cual se añaden entradas de menú, que son objetos de la clase `JMenu`. A cada objeto `JMenu` se le añaden tantos objetos de la clase `JMenuItem` como acciones tiene asociadas esa entrada del menú.

Las principales características de esta clase son:

- Esta clase implementa un único menú (`private JMenu file`), que se muestra con la etiqueta “Fichero”.
- El menú cuenta con un único item (`JMenuItem salir`), etiquetado con “Salir” y que se asocia a la acción `ItemSalir`. Esta es la cadena que recibirá el controlador cuando el usuario seleccione este item del menú.
- La clase cuenta con un método `setActionListener` que permite asociar un controlador al menú. Fíjate como a este método se le pasa como único argumento una referencia a un objeto de tipo

ActionListener, que es utilizado en la línea 25. Indicar que en Swing es necesario registrar el oyente(listener) para cada JMenuItem del JMenu. El método addActionListener permite registrar un objeto JMenuItem con el oyente (una instancia de la clase PeliculaControllerActionListener)

Listado 8: La clase PeliculaMenu.java que implementa la barra de menú de nuestra aplicación.

```

1
2 package es.uv.eu.aplicacion4;
3
4 import java.awt.event.ActionListener;
5 import javax.swing.JMenu;
6 import javax.swing.JMenuBar;
7 import javax.swing.JMenuItem;
8
9 *
10
11 @author Entornos de Usuario
12 /
13 public class PeliculaMenu extends JMenuBar {
14     private JMenu file;
15     private JMenuItem salir;
16
17     public PeliculaMenu() {
18         file = new JMenu("Archivo");
19         salir = new JMenuItem("Salir");
20         salir.setActionCommand("ItemSalir");
21         file.add(salir);
22         this.add(file);
23     }
24
25     public void setActionListener(ActionListener actionListener) {
26         salir.addActionListener(actionListener);
27     }
28 }

```

5.2 La nueva clase PeliculaView

Ahora es necesario modificar nuestra clase PeliculaView para que añada nuestro menú como nuevo elemento gráfico. En el listado 9 se muestra el código de la nueva clase.

Las características básicas de esta clase son:

- Añadimos un nuevo atributo private PeliculaMenu peliculaMenu que implementa el menú.
- Añadimos el menú al JFrame mediante el método setJMenuBar.
- Modificamos el método setActionListener para extender la llamada

Listado 9: La clase PeliculaView.java ampliada con el menú de la aplicación.

```

1
2 package es.uv.eu.aplicacion5;
3
4 import java.awt.BorderLayout;
5 import java.awt.Font;
6 import java.awt.event.ActionListener;
7 import javax.swing.JButton;
8 import javax.swing.JFrame;
9 import javax.swing.JLabel;
10
11 *

```

```

12
13 @author Entornos de Usuario
14 /
15 public class PeliculaView extends JFrame {
16
17     private PeliculaModel peliculas;
18     private ListaPanel listaPanel;
19     private JButton exit;
20     private PeliculaMenu peliculaMenu;
21
22     public PeliculaView(PeliculaModel peliculas) {
23         // Invoca al constructor de la superclase Frame
24         super("Aplicacion 1: Lista de peliculas");
25
26         this.peliculas = peliculas;
27         this.setLayout(new BorderLayout());
28
29         // Tamano del Frame
30         this.setSize(700,400);
31
32         // Menu de la aplicacion
33         peliculaMenu = new PeliculaMenu();
34         this.setJMenuBar(peliculaMenu);
35
36         // Cabecera identificativa de la aplicacion
37         JLabel label = new JLabel("Lista de películas:");
38         label.setFont(new Font("Sans", Font.BOLD, 14));
39
40         // El panel con la lista de peliculas
41         listaPanel = new ListaPanel(peliculas);
42         listaPanel.mostrar();
43
44         // El boton de salida se anyade en la aplicacion 2
45         exit = new JButton("Salir");
46         exit.setActionCommand("buttonExit");
47
48         // ñAade todos los componentes al frame
49         this.add(label, BorderLayout.NORTH);
50         this.add(listaPanel, BorderLayout.CENTER);
51         this.add(exit, BorderLayout.SOUTH);
52
53
54         this.setVisible(true);
55     }
56     public void setActionListener(ActionListener actionListener) {
57         exit.addActionListener(actionListener);
58         peliculaMenu.setActionListener(actionListener);
59     }

```

5.3 La clase PeliculaController modificada

Ahora modificaremos la clase `PeliculaController` para que responda al nuevo evento que puede generar el usuario. La única modificación (ver listado 10) es que hemos añadido en el `switch` de la clase `PeliculaControllerActionListener` la entrada `case 'ItemSalir'` que responde a la selección del ítem del menú **Archivo** → **Salir** que acabamos de añadir a la aplicación.

Listado 10: La clase `PeliculaController.java` modificada para responder a los eventos del menú.

```

1 package es.uv.eu.aplicacion1;
2
3 import java.awt.event.ActionEvent;

```

```

4 import java.awt.event.ActionListener;
5 import java.awt.event.WindowAdapter;
6 import java.awt.event.WindowEvent;
7
8 /**
9  *
10  * @author Entornos de usuario
11  */
12 public class PeliculaController {
13     private PeliculaModel model;
14     private PeliculaView view;
15
16     public PeliculaController (PeliculaModel model, PeliculaView view) {
17         this .model = model;
18         this .view = view;
19
20         view.addWindowListener(new PeliculaControllerWindowListener());
21         view.addActionListener(new PeliculaControllerActionListener ());
22     }
23
24     // Clases empotradas
25     class PeliculaControllerWindowListener extends WindowAdapter {
26         @Override
27         public void windowClosing(WindowEvent e) {
28             System.out.println(" PeliculaController : Cerrar ventana.");
29             System.exit(0);
30         }
31     }
32     class PeliculaControllerActionListener implements ActionListener {
33         @Override
34         public void actionPerformed(ActionEvent ae) {
35             String command = ae.getActionCommand();
36             switch (command) {
37                 case "buttonExit":
38                     System.out.println(" PeliculaController : Boton 'Salir' .");
39                     System.exit(0);
40                     break;
41                 case "ItemSalir":
42                     System.out.println(" PeliculaController : Menu 'Salir' .");
43                     System.exit(0);
44                     break;
45                 default :
46                     System.out.println(" PeliculaController : Comando '" + command + "' no reconocido.");
47                     break;
48             }
49         }
50     }
51 }

```

6 Ejercicio E

Este ejercicio será desarrollado íntegramente por ti y tiene como objetivo que pongas en práctica todos los conocimientos adquiridos y que adquieras cierta capacidad para buscar y aplicar las funciones de la API Swing de Java.

La propuesta es que, partiendo del último código de la aplicación (Ejercicio D), añadas nueva funcionalidad al programa. Esta nueva aplicación deberá contar con dos menús adicionales en la barra del menú, un menú **Ayuda** y un menú **Ventana**. El menú **Ayuda**, tendrá un único ítem **Acerca de...**, que generará una nueva ventana con información de la aplicación y sus desarrolladores, de modo similar a como se muestra en la figura 4. El otro menú, menú **Ventana**, consta de dos ítems: **Maximizar** y **Tamaño por defecto**, tal

como se muestra en la figura 5. El primer ítem permite ajustar la ventana de la aplicación al tamaño de la pantalla del ordenador. El segundo ítem permite que la ventana de la aplicación se ajuste al tamaño inicial que tiene la ventana por defecto (700x400) cuando iniciamos la aplicación.

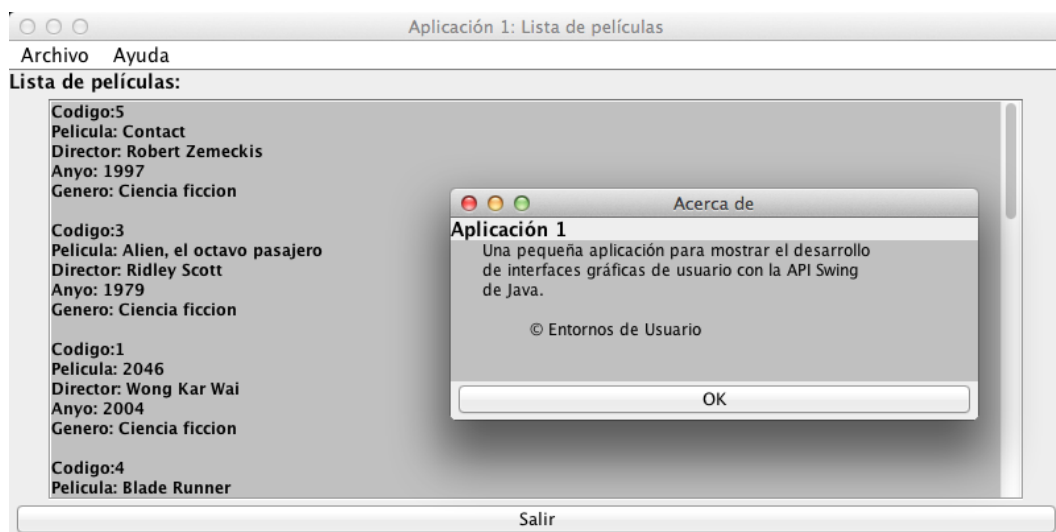


Figura 4: Salida de consola producida por la aplicación `Aplicacion5` que tienes que desarrollar. Personaliza el mensaje mostrado en la ventana con tus datos.

Para realizar la aplicación, pueden ser útiles las siguientes indicaciones sobre las aplicaciones Java-Swing:

- Deberás modificar la clase `PeliculaMenu` para que incorpore las dos nuevas entrada del menú, `Ayuda` y `Ventana` y los ítems de asociados a cada uno de ellos. Además deberás modificar el método `setActionListener` para que registre los ítems con el oyente (listener).
- También deberás modificar `PeliculaController` para que responda al evento generado por los ítems de los diferentes menús. Para ello deberás añadir varios `case` al `switch`.
- Será necesario crear una nueva clase (`AcercaView`) que, como la vista principal `PeliculaView`, deberá derivar de la clase `Frame` y que mostrará la información de la aplicación. Observa que esta clase tendrá una estructura muy similar a la de la clase `PeliculaView`, ya que contará con muchos de los elementos que hemos visto en esta práctica (*layouts, panels, buttons*, etc...).
- Para obtener el tamaño de la pantalla, dato necesario para tratar el evento que se produce a pulsar sobre el ítem `Maximizar`, será necesario consultar los método de la clase abstracta `Toolkit`.

Nota: Consulta en la API `javax.swing` accesible desde <http://docs.oracle.com/javase/7/docs/api/index.html>, para consultar los métodos y clases que te permitan desarrollar este ejercicio.

6.1 Entrega

Esta práctica se realizará en una sesión. La entrega del EJERCICIO E, debidamente documentado, se realizará en Aula Virtual antes del comienzo de la siguiente sesión de prácticas.

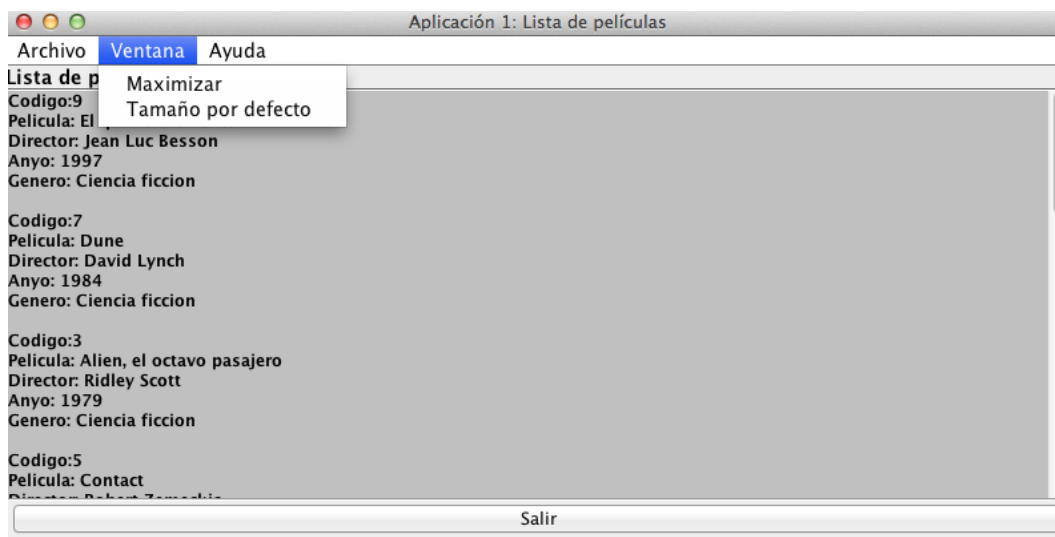


Figura 5: Salida de consola producida por la aplicación `Aplicacion5` que tienes que desarrollar. Se muestra el menú ventana y los ítems que contiene