UNIVERSITAT DE BARCELONA

Ingeniería Informática

Programación II

Reproductor Multimedia

Práctica 4

Grupo C

Nombre	NIUB
Manuel Ernesto Martínez Martín	20081703
Aarón Peruga Ortiga	20026941

Tabla de contenidos

Introducción

En esta última práctica de laboratorio de Programación II hemos diseñado una GUI (Graphical User Inteface) con la única finalidad de que haga de la vista de nuestro proyecto.

La GUI ha sido implementado gracias a la biblioteca de JAVA “swing” que nos otorga una serie de elementos para cubrir todas las funciones que el reproductor necesita.

Al final, la GUI que hemos diseñado nos sirve para poder controlar nuestro reproductor multimedia que hemos ido desarrollando en las anteriores prácticas.

Análisis

Nuestro reproductor está formado por todas las clases de la práctica anterior excepto las pertenecientes a la vista y estas tres clases que serán la vista nueva:

- **AplicacioUB4**: es la ventana principal de nuestro reproductor, contiene todos los elementos para poder realizar las funciones principales de nuestro reproductor como crear un álbum, añadir un fichero, eliminar un álbum.
- **FrmAfegirFitxerMultimedia:** es un ventana que aparece cuando pulsamos la opción “Afegir Fitxer”, para rellenar la información.

  En esta ventana podremos elegir entre añadir un archivo de audio o de video y dependiendo de la opción que escojamos la ventana cambiará para mostrarnos la información correspondiente para añadirlo correctamente.
- **FrmSobre:** es la ventana de acerca de del programa.

Desarrollo

Como hemos dicho en [Análisis], en este proyecto hemos desarrollado tres clases que serán nuestras ventanas principales:

JFrame AplicacioUB4

Es nuestra ventana principal en ella tenemos:

- **JMenuBar mbMenu:** barra de menú.
- **JMenu mMenu:** opción principal del menú.
- **JMenu mDades:** opción del menú referente a Dades (Guardar/Cargar).
- **JMenuItem miCarregar:** botón del menú para cargar datos.
- **JMenuItem miGuardar:** botón del menú para guardar datos.
- **JMenuItem miSortir:** botón del menú para salir del programa.
- **JMenu mPreferencies:** opción del menú referente a ajustes de reproducción (aleatoria/continua).
- **JCheckBoxMenuItem cbmiRepAleatoria:** activa/desactiva la reproducción aleatoria.
- **JCheckBoxMenuItem cbmiRepContinua:** activa/desactiva la reproducción continua.
- **JMenu mInfo:** opción del menú referente a información.
- **JMenuItem miSobre:** botón del menú para abrir la ventana de acerca de.
- **JLabel etBiblioteca:** etiqueta de biblioteca (identificativa).
- **JList<String> lstBiblioteca:** lista de ficheros de la biblioteca.
- **JScrollPane scpBiblioteca:** panel de scroll para la lista de ficheros de la biblioteca.
- **JButton btnAfegirFitxerBiblioteca:** abrirá la ventana FrmAfegirFitxerMultimedia para poder añadir el archivo que queramos.
- **JButton btnEliminarFitxerBiblioteca:** elimina uno o varios ficheros de la biblioteca.
- **JButton btnReproduirFitxerBiblioteca:** reproduce un fichero de la biblioteca.
- **JButton btnReproduirBiblioteca:** reproduce la biblioteca entera.
- **JLabel etAlbums:** etiqueta de albums (identificativa).
- **JComboBox<String> cmbAlbums:** listado de álbumes.
- **JButton btnCrearAlbum:** a través de JOptionPane creamos un álbum con dos ventanas que nos pedirán título y capacidad.

- **JButton btnEliminarAlbum:** elimina el álbum seleccionado.
- **JButton btnReproduirAlbum:** reproduce el álbum seleccionado.
- **JLabel etAlbum:** etiqueta de álbum (identificativa).
- **JList<String> lstAlbum:** lista de ficheros del álbum.
- **JScrollPane scpAlbum:** panel de scroll para la lista de ficheros del álbum.
- **JButton btnEliminarFitxerAlbum:** elimina uno o varios ficheros del álbum seleccionado.
- **JButton btnAfegirFitxerAlbum:** añade uno o varios archivos al álbum seleccionado.
- **JButton btnReproduirFitxerAlbum:** reproduce un fichero del álbum.
- **JButton btnReempren:** continúa la reproducción.
- **JButton btnAtura:** para la reproducción.
- **JButton btnPause:** pausa la reproducción.
- **JButton btnSalta:** salta al siguiente fichero reproducible en una reproducción activa.

JDialog FrmAfegirFitxerBiblioteca

Ventana formulario para recoger los datos del fichero que se desea añadir.

- **JButton btnObrirFitxer:** usa JFileChooser para seleccionar fichero.
- **JButton btnObrirImatge:** usa JFileChooser para seleccionar fichero de imagen.
- **ButtonGroup btngpAudioVideo:** contenedor de los radio botones (audio/video).
- **JRadioButton rbtnAudio:** radio botón para modo añadir audio.
- **JRadioButton rbtnVideo:** radio botón para modo añadir video.
- **JLabel etCami:** etiqueta camino del fichero.
- **JTextField textCami:** campo para introducir el camino del fichero.
- **JLabel etNom:** etiqueta nombre de fichero.
- **JTextField textNom:** campo para introducir nombre de fichero.
- **JLabel etCodec:** etiqueta codec de fichero.
- **JTextField textCodec:** campo para introducir el codec del fichero.
- **JLabel etDurada:** etiqueta duración de fichero.
- **JTextField textDurada:** campo para introducir la duración del fichero.
- **JLabel etFpsKbps:** etiqueta fps en video / kbps en audio.
- **JTextField textFpsKbps:** campo para introducir fps de video o kbps de audio.
- **JPanel pnlVideo:** panel de componentes de video.
- **JLabel etAlcada:** etiqueta altura de video.
- **JTextField textAlcada:** campo para introducir altura de video.
- **JLabel etAmplada:** etiqueta ancho de video.
- **JTextField textAmplada:** campo para introducir ancho de video.
- **JPanel pnlAudioImatge:** panel de componentes de audio.
- **JLabel etImatge:** etiqueta fichero imagen del audio.
- **JTextField textCamiImatge:** campo para introducir el camino de la imagen del audio.
- **JButton btnAcceptar:** guarda si es posible el fichero.
- **JButton btnCancelar:** cancela añadir un fichero.

Después de rellenar en **FrmAfegirFitxerMultimedia** los datos para añadir el archivo a la biblioteca **tenemos un evento al cerrar la ventana para que refresque el JList de la biblioteca**, **Solo si se ha añadido** (con un atributo booleano).

```
FrmAfegirFitxerMultimedia form = new FrmAfegirFitxerMultimedia(this, true);
form.addWindowListener(new WindowAdapter() {
    @Override
    public void windowClosed(WindowEvent ev) {
        if (form.isAfegit()) {
            refreshBiblioteca();
        }
    }
});
form.setVisible(true);
```

Imagen: evento al cerrar FrmAfegirFitxerMultimedia.

Dependiendo de si marcamos Audio o Video, **ocultaremos y mostraremos opciones diferentes de la ventana**, **además de algunas otras cosas** como limpiar el texto o renombrar etiquetas (por defecto el formulario es para Audio).

JDialog FrmSobre

Ventana de acerca de, de la aplicación:

- **JButton btnTornar:** cierra la ventana de acerca de.
- **JLabel etAutor1:** etiqueta del nombre de un autor.
- **JLabel etAutor2:** etiqueta del nombre de un autor.
- **JLabel etAutors:** etiqueta autores.
- **JLabel etCopyRight:** etiqueta de copyright.
- **JLabel etGPLV3:** logo de la licencia gplv3.
- **JLabel etLlicencia:** etiqueta licencia.
- **JLabel etLogoUb:** logo de la Universidad de Barcelona.
- **JLabel etNombrePrograma:** etiqueta del nombre del programa.
- **JLabel etVersion:** etiqueta de versión.
- **JLabel etVersionNumber:** etiqueta del número de versión del programa.
- **JPanel panelCopy:** panel donde está la etiqueta del copyright.

Cuestiones planteadas

1. En un primer momento teníamos dudas de a la hora de hacer modificaciones en los datos de que manera mostrarlos:
   - Cada vez que modifiquemos los datos hacer setModel() en JList/JComboBox para actualizar la vista de los datos cambiados.

   - Usar el mismo Model modificando su contenido, por ejemplo si en el JList del álbum seleccionado se borra un archivo, entonces llamamos al controlador para que nos de una lista de los archivos que hay en ese álbum, acto seguido hacemos removeAll() del Model de lstAlbum y añadimos todos los archivos de la lista consultada a ese mismo Model.

   Hemos probado ambas, y para como está la práctica nos hemos decantado por hacer un setModel() a cada componente modificado (JList, JComboBox).

2. Referente a los formularios, queríamos que estuvieran limitados en su contenido al tipo de dato que se necesita, podríamos haber usado otro tipo de contenedor, pero hemos decidido usar JTextField y hacer eventos KeyTyped, dónde evaluamos la escritura así:

```
/**
 * Controla el evento para que acepte solo números ints válidos el
 * JTextField
 *
 * @param tf
 * @param evt
 */
public static void onlyIntNumbers(JTextField tf, KeyEvent evt) {
    char vChar = evt.getKeyChar();
    if (!(Character.isDigit(vChar) || (vChar == KeyEvent.VK_BACK_SPACE)
            || (vChar == KeyEvent.VK_DELETE))) {
        evt.consume();
    }
    String txt = tf.getText();
    try {
        int screen = Integer.parseInt(txt);
    } catch (NumberFormatException ne) {
        if (txt.isEmpty()) {
            txt = "0";
        }
        if (Long.parseLong(txt) > Integer.MAX_VALUE) {
            evt.consume();
        }
    }
}
```

Imagen: código de verificación para número entero en un JTextField

```java
/**
 * Controla el evento para que acepte solo números floats válidos el
 * JTextField
 *
 * @param tf
 * @param evt
 */
public static void onlyFloatNumbers(JTextField tf, KeyEvent evt) {
    char vChar = evt.getKeyChar();
    if (!(Character.isDigit(vChar) || (vChar == KeyEvent.VK_BACK_SPACE)
            || (vChar == KeyEvent.VK_DELETE) || (vChar == KeyEvent.VK_PERIOD && !tf.getText().contains(".") && !tf.getText().isEmpty()))) {
        evt.consume();
    }
    String txt = tf.getText();
    try {
        float screen = Float.parseFloat(txt);
    } catch (NumberFormatException ne) {
        if (txt.isEmpty()) {
            txt = "0.0";
        }
        if (Double.parseDouble(txt) > Float.MAX_VALUE) {
            evt.consume();
        }

    }
}
```

Imagen: código de verificación para número float en un JTextField

3. Al final hemos decidido mantener IniciadorAplicacioUB solo para el main()

Reutilización de la práctica 3

Clases reaprovechadas

Hemos reutilizado todas las clases de la práctica anterior excepto las referentes a la vista

Clases en el diagrama [UML]

Cambios sobre las clases reutilizadas

- En **Controlador** y **Dades** hemos añadido dos métodos a cada uno para obtener el toString() del fichero requerido ya sea de un álbum o de la biblioteca, para así al dar dos clicks en un elemento del JList se nos muestre la información completa del fichero.
    - **Controlador:**

```
/**
 * Retorna la información del fichero de la biblioteca a través de Dades
 *
 * @param id
 * @return String
 */
public String infoFitxer(int id) {
    return dades.infoFitxer(id);
}

/**
 * Retorna la información del fichero del album a través de Dades
 *
 * @param titolAlbum
 * @param id
 * @return String
 * @throws AplicacioException
 */
public String infoFitxer(String titolAlbum, int id) throws AplicacioException {
    return dades.infoFitxer(titolAlbum, id);
}
```

- **Dades:**

```
/**
 * Retorna la información del fichero de la biblioteca
 *
 * @param id
 * @return String
 */
public String infoFitxer(int id) {
    return biblioteca.getAt(id).toString();
}

/**
 * Retorna la información del fichero del album
 *
 * @param titolAlbum
 * @param id
 * @return String
 * @throws AplicacioException
 */
public String infoFitxer(String titolAlbum, int id) throws AplicacioException {
    try {
        AlbumFitxersMultimedia album = getAlbumByTitle(titolAlbum);
        if (album.getSize() == 0) {
            throw new AplicacioException("No hi ha fitxers en aquest album");
        }
        FitxerMultimedia file = (FitxerMultimedia) album.getAt(id);
        return file.toString();
    } catch (IndexOutOfBoundsException io) {
        throw new AplicacioException("Id de fitxer incorrecte");
    }
}
```

- En **FitxerMultimedia**, **FitxerReproduible**, **Audio** y **Video** hemos modificado sus métodos toString() lo dicho en el punto anterior de **Controlador** y **Dades**, de manera que mostramos un JOptionPane al dar dos clicks al elemento con su información así:

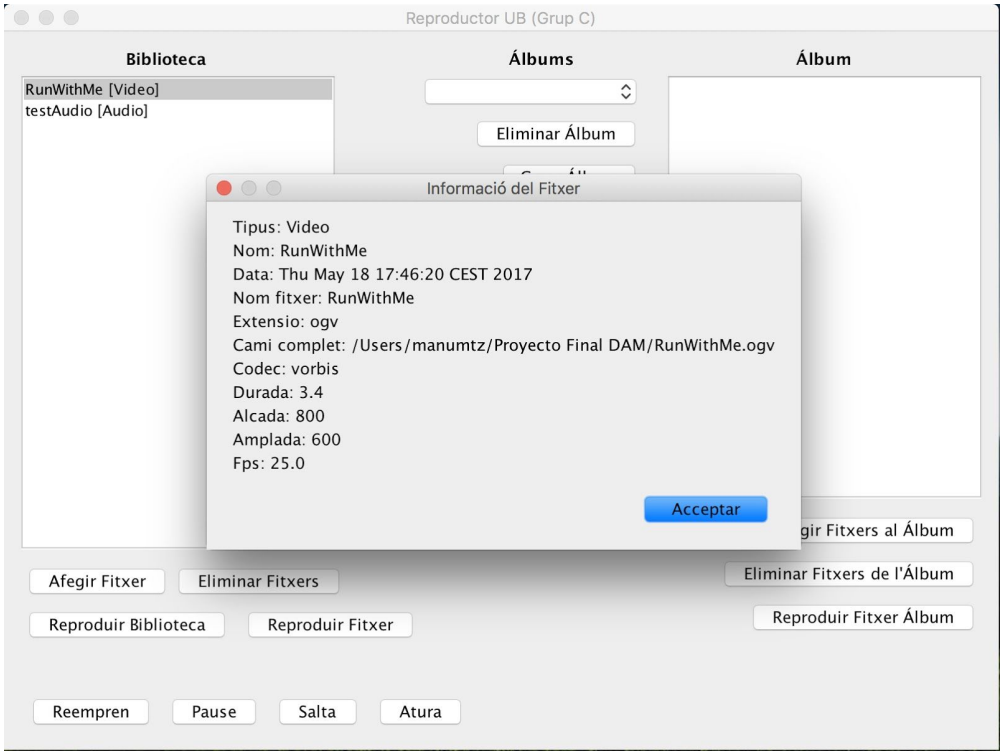


Imagen: información de fichero de video en Macos High Sierra.

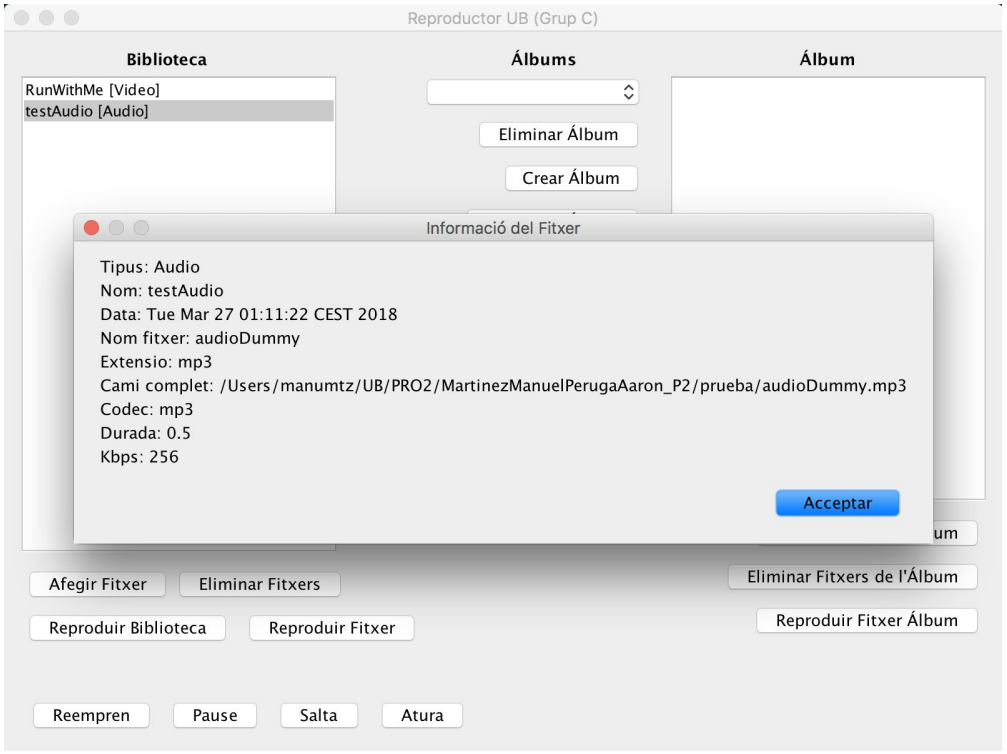


Imagen: información de fichero de audio en Macos High Sierra.

Modelo de delegación de eventos

Botón de reproducción de la Biblioteca

El botón tiene añadido un **Listener** para escuchar cuando se realiza una acción, que en nuestro caso es un **ActionPerformed**, el generador de formularios de Netbeans lo genera automáticamente y además te genera un método nuevo donde permite escritura, el original lo deja bloqueado.

```
btnReproduirBiblioteca.setText("Reproduir Biblioteca");
btnReproduirBiblioteca.setFocusPainted(false);
btnReproduirBiblioteca.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        btnReproduirBibliotecaActionPerformed(evt);
    }
});
```

El método dónde podemos escribir:

```
private void btnReproduirBibliotecaActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        ctrl.reproduirCarpeta();
    } catch (AplicacioException ex) {
        JOptionPane.showMessageDialog(this, ex.getMessage(), "Error", JOptionPane.PLAIN_MESSAGE, ERROR_IMG);
    }
}
```

En un Listener podemos tener varios tipos de eventos, pero en el caso del botón reproducir solo queremos el de hacer click que vendría a ser ActionPerformed.

Para simplificar, dónde se produce el evento es en el botón, como este tiene un Listener registrado detectamos que hemos pulsado el botón y por consiguiente ejecutamos la acción requerida.

Tipos de eventos usados en el código

- **AplicacioUB4**
  - JMenuItem → **ActionPerformed** (Acciones del menú) - **ActionEvent**
  - JComboBox → **ActionPerformed** (selección de álbum) - **ActionEvent**
  - JButton → **ActionPerformed** (diferentes acciones de botones) - **ActionEvent**
  - JCheckBoxMenuItem → **ActionPerformed** (aleatoria/continua) - **ActionEvent**
  - JFrame form → **MouseClicked** (para des-seleccionar los archivos de los JList) - **MouseEvent**
  - JList → **MouseClicked** (para mostrar información del fichero con doble click) - **MouseEvent**
  - JDialog → **WindowClosed** (para actualizar el JList de la biblioteca al añadir) - **WindowEvent**
- **FrmAfegirFitxerMultimedia**
  - JButton → **ActionPerformed** (añadir fichero o cancelar) - **ActionEvent**
  - JRadioButton → **ActionPerformed** (marcar un radio botón) - **ActionEvent**
  - JTextField → **KeyTyped** (para comprobar lo escrito) - **KeyEvent**
- **FrmSobre**
  - JButton → **ActionPerformed** (para cerrar ventana) - **ActionEvent**

Observaciones generales

Creemos que **debería haber** un **Listener** en Dades **o un patrón Observer** de tal manera que **se actualice automáticamente la información en la vista después de cada operación**.

Reproductor no debería ser un atributo de FitxerReproduible, puesto que ya está en controlador.

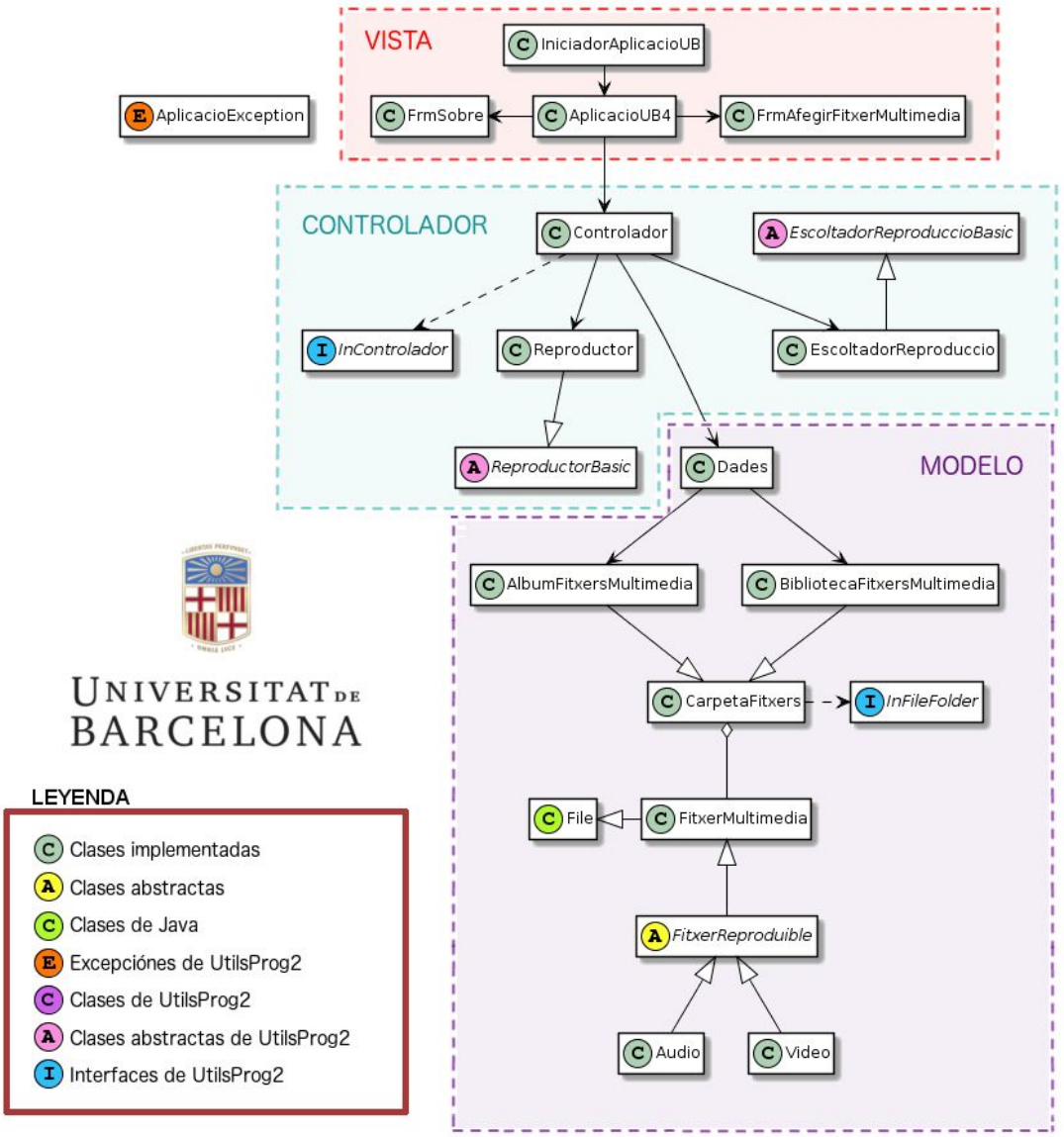
Debería haber un Observador con Escuchador o Listener para saber si hay reproducción en curso para tener activados o desactivados los botones: pause, atura, reempren y salta.

Hemos visto que **en FrmAfegirFitxerMultimedia podíamos aprovechar KBS y FPS** para crear Audio o Video, de esta manera tendríamos un JLabel y un jTextField que serviría para ambos, solo que depende de si es audio o video pondrá cosas diferentes y aceptara ints o floats dependiendo.

Los iconos que vienen por defecto en el JOptionPane al menos en mac eran poco informativos, así que decidimos poner otros.

Hemos decidido después de probar diferentes LookAndFeels dejar el **aspecto del sistema**.

Diagrama UML de clases simple



Pruebas realizadas y resultados obtenidos

Reproducción

Material de prueba: 7 archivos (EN ESE ORDEN):

1 archivo guardado como audio o video que no sea ninguno (ej: un .txt)
1 canción con carátula.
1 canción con carátula que no exista.
1 archivo guardado como audio o video que no sea ninguno (ej: un .txt)
1 canción sin caratula.
1 video.
1 archivo guardado como audio o video que no sea ninguno (ej: un .txt)

- **Prueba 1**: reproducción de los 7 archivos normal (sin cíclica ni aleatorio).
- **Prueba 2**: reproducción de los 7 archivos NO cíclica pero SI aleatorio.
- **Prueba 3**: reproducción de los 7 archivos SI cíclica y NO aleatorio.
- **Prueba 4**: reproducción de los 7 archivos SI cíclica y SI aleatorio.

Borrado de los 3 archivos (no reproducibles) y quedarán 4 archivos reproducibles

- **Prueba 5**: reproducción de los 4 archivos normal (sin cíclica ni aleatorio).
- **Prueba 6**: reproducción de los 4 archivos NO cíclica pero SI aleatorio.
- **Prueba 7**: reproducción de los 4 archivos SI cíclica y NO aleatorio.
- **Prueba 8**: reproducción de los 4 archivos SI cíclica y SI aleatorio.

Uso solo de archivos no válidos para reproducir

- **Prueba 9**: reproducción de los 3 archivos normal (sin cíclica ni aleatorio).
- **Prueba 10**: reproducción de los 3 archivos NO cíclica pero SI aleatorio.
- **Prueba 11**: reproducción de los 2 archivos SI cíclica y SI aleatorio.

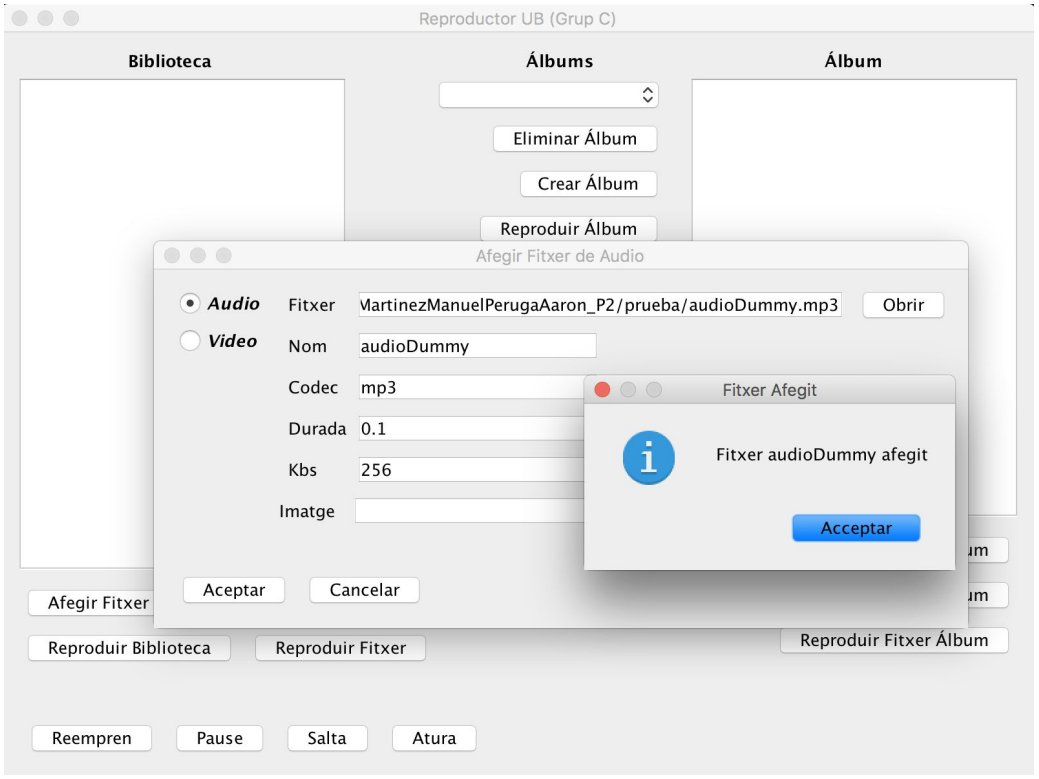
Borrado de varios archivos

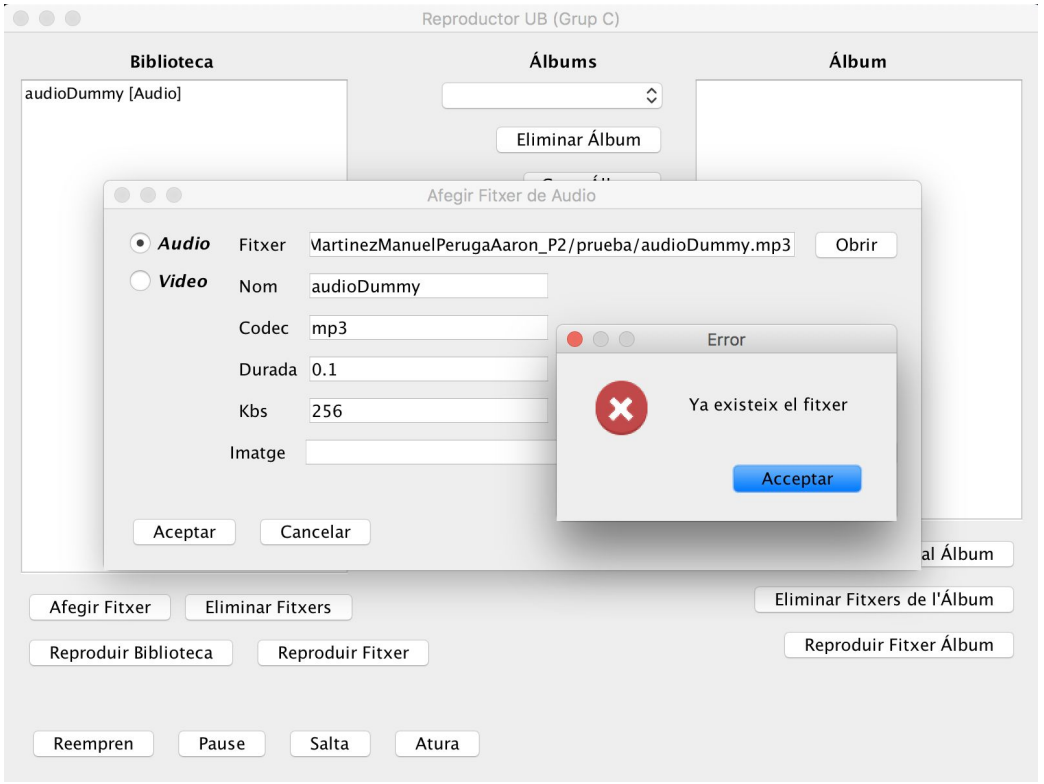
Hemos comprobado que se borran varios archivos seleccionados desde Biblioteca o desde un Álbum.

Añadir Fichero

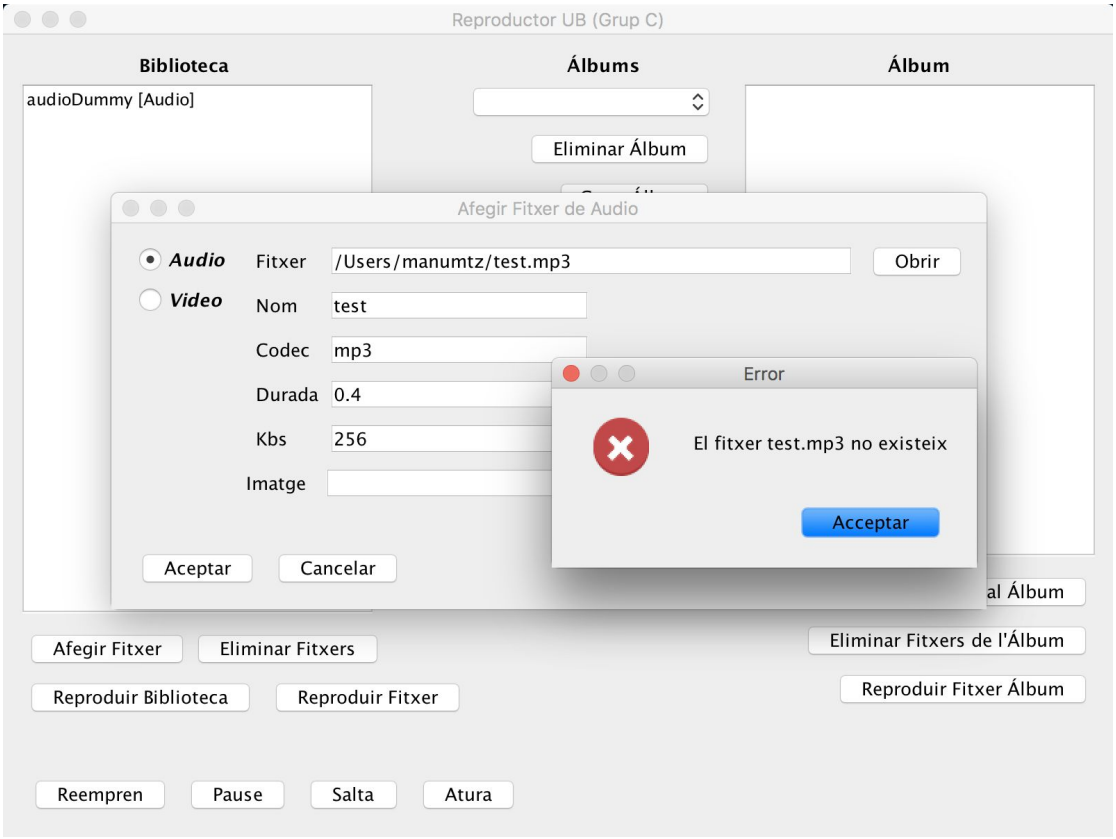
Todos los campos son obligatorios excepto el camiImatge de Audio.

Archivos iguales a la biblioteca

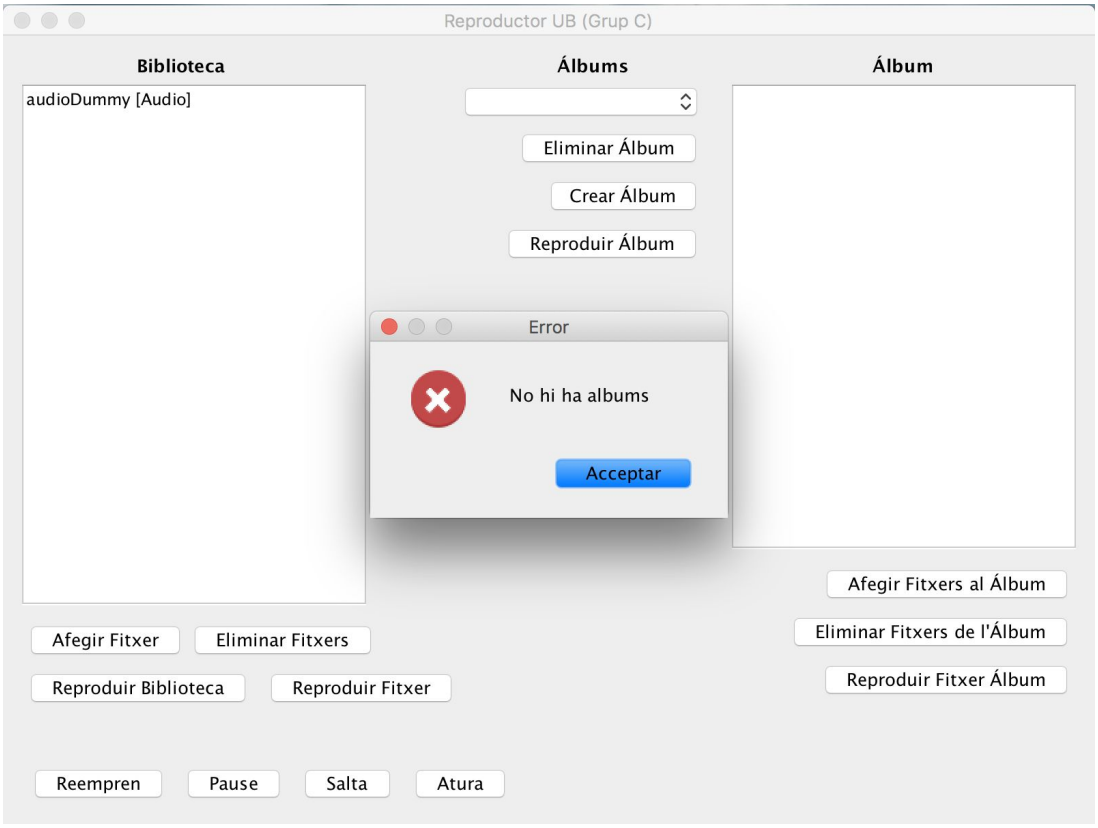




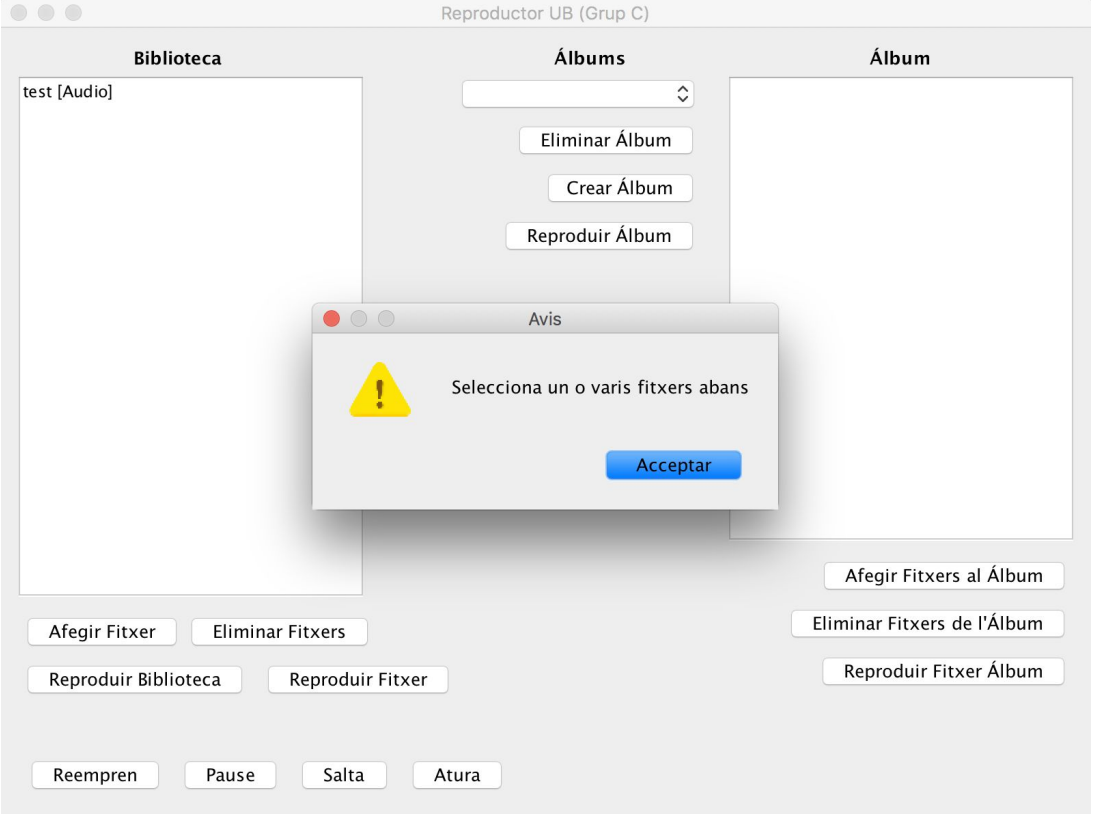
Archivo que no existe a la biblioteca



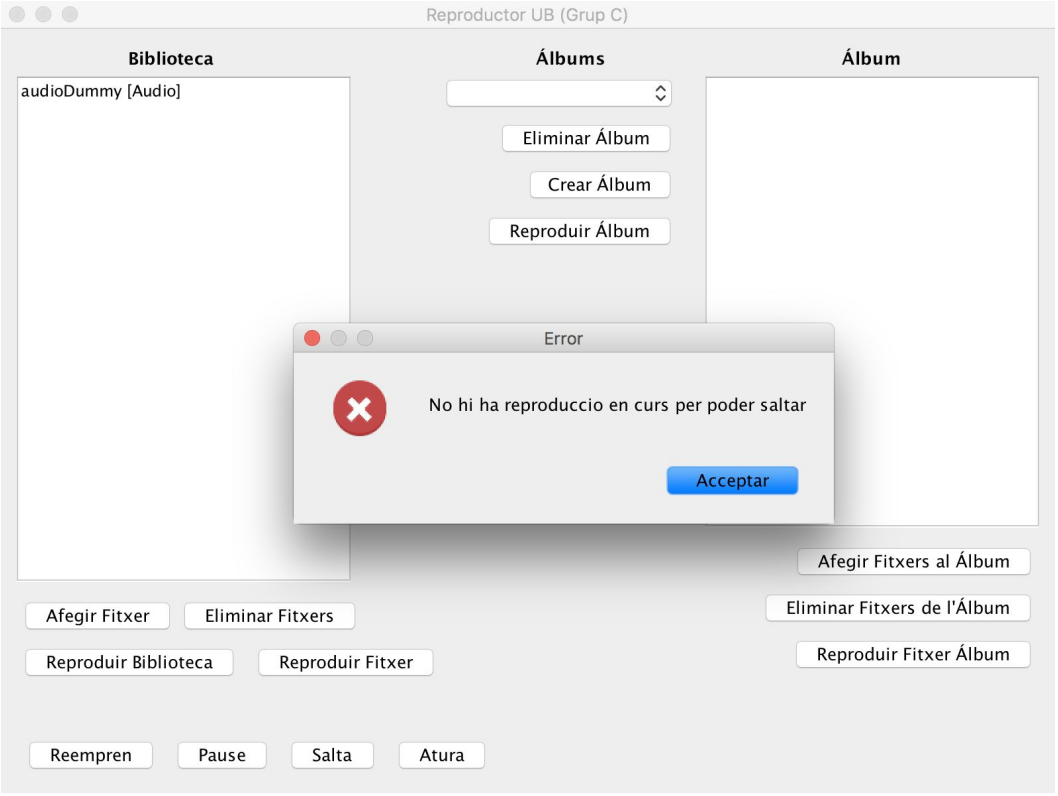
Borrar un álbum que no existe



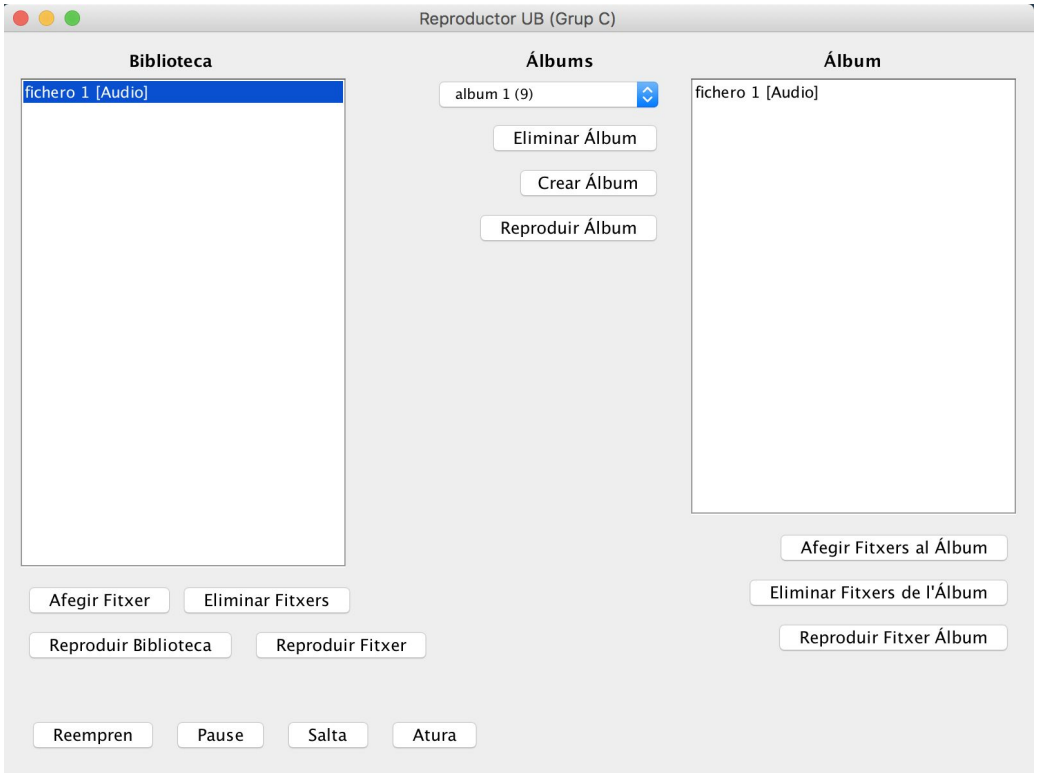
Borrar sin seleccionar fichero en biblioteca (en un álbum es igual)

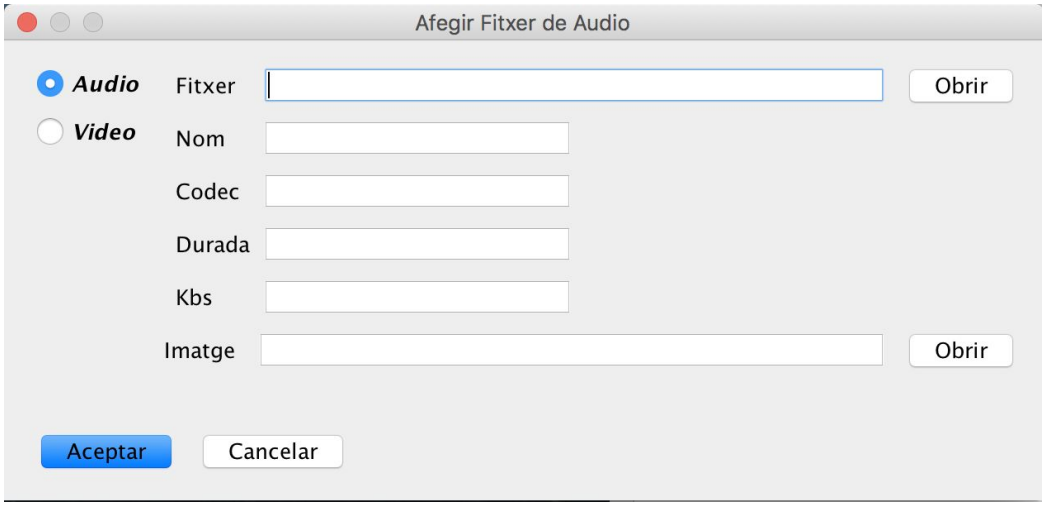


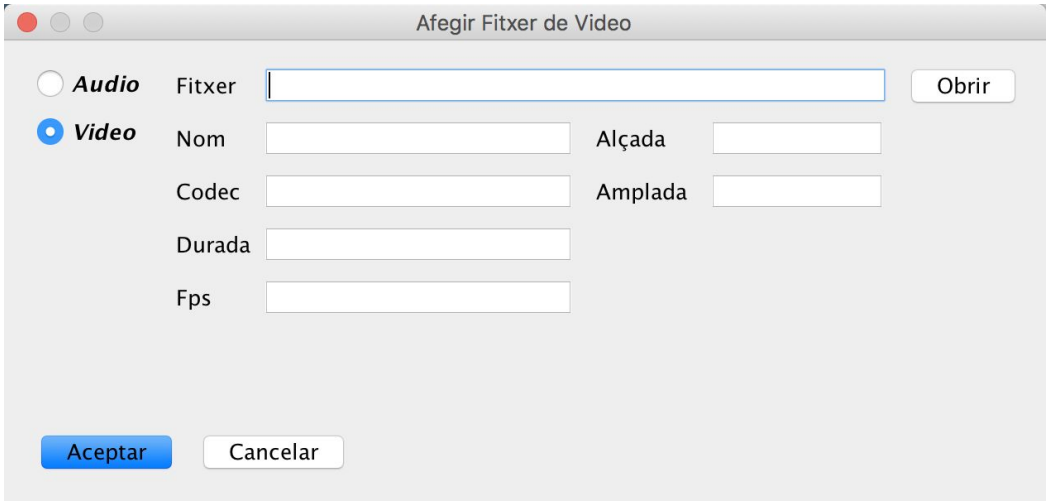
Pausa o Atura o Salta o Reempren sin tener reproducción activa



Vista Previa






Afegir Fitxer de Video
Audio
Video
Fitxer
Obrir
Nom
Alçada
Codec
Amplada
Durada
Fps
Aceptar
Cancelar


Reproductor UB (Grup C)
Versión: 4.0
Autors: Manuel Martínez Martín
Aarón Peruga Ortiga
Llicència:
GPL
Free Software
Copyright 2018 Universitat de Barcelona
Tornar

En Mac la barra de menú se encuentra arriba