REPRODUCTOR DE VIDEO

Documentación Técnica

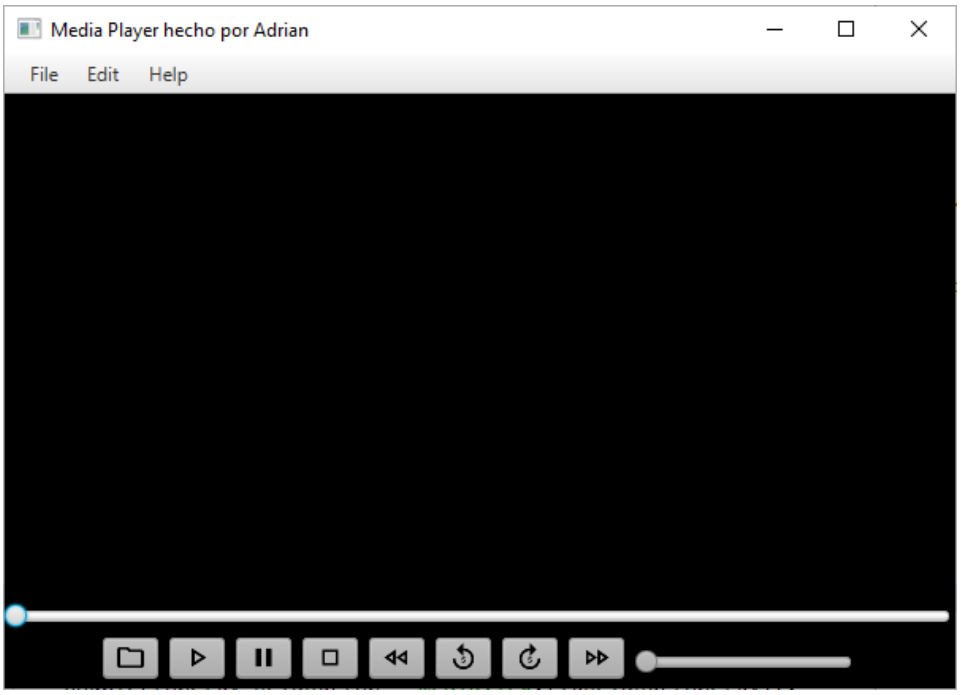
Adrián Martínez Medina

18 de marzo de 2022

Contenido

Introducción

El proyecto final de java consiste en un reproductor de vídeo desarrollado con el IDE Netbeans 12.4 junto con el programa JavaFX Scene Builder, el cual nos permitirá diseñar de manera gráfica la aplicación, ya que está desarrollada con componentes JavaFX.



Como cualquier aplicación que sea un reproductor de vídeo, cuenta con las funcionalidades de abrir vídeo, play, pause, stop, ralentizar, acelerar, rebobinar e ir hacia adelante, además de una barra de volumen y la barra de progreso del vídeo. Las funcionalidades son activables tanto desde sus respectivos botones como a través de combinaciones de teclado. Todas estas características de la aplicación serán explicadas más adelante en su apartado.

Además de las funcionalidades mencionadas, la aplicación cuenta con un menú bar, desde el que podemos abrir vídeos recientes, así como con un “Acerca de” que muestra por encima información de la aplicación.

Conceptos previos al programa

Oracle Netbeans IDE

Tal y como se ha mencionado en la introducción del documento, nuestra aplicación ha sido desarrollada totalmente en el IDE Oracle Netbeans, más concretamente con su versión 12.4, lo cual es un dato importante debido a que, si se desea realizar en un futuro cualquier modificación, y ésta se lleva a cabo en una versión diferente a la que se usó durante su desarrollo inicial, podría provocar múltiples errores en el proyecto.

Antes de explicar y analizar las principales características de Oracle Netbeans IDE, debemos definir el concepto de “IDE”.

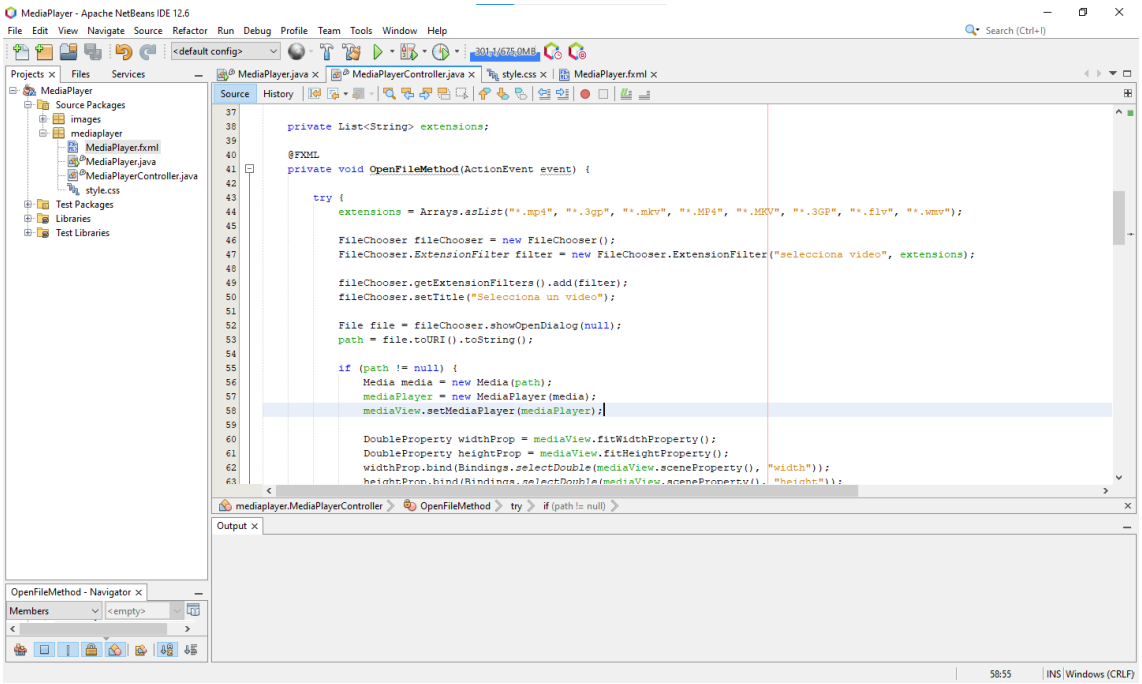
Saber qué son las IDE y para qué se utilizan es fundamental si se quiere programar aplicaciones o desarrollar páginas web. Un IDE es una herramienta básica para que un programador pueda trabajar en un marco amigable que le permita realizar sus tareas de forma mucho más ágil y eficiente.

Las características del IDE son diversas y suelen incluir la capacidad multiplataforma (para el desarrollo para distintos sistemas), el soporte para diferentes lenguajes de programación, la integración con un sistema para control de versiones, reconocimiento de sintaxis (para evitar errores al escribir código y acelerar el proceso de codificación), la integración con entornos de trabajo, capacidad de depuración de código, soporte para múltiples idiomas y posibilidad de importar y exportar proyectos.

Un IDE está compuesto principalmente por los siguientes componentes:

- Editor de texto.
- Intérprete y compilador.
- Herramientas para la automatización de tareas.
- Depurador de código.
- Sistema de control de versiones (la mayoría).
- Sistema de diseño GUI o interfaz gráfica de usuario (botones, listas desplegables, etc.).

Una vez definido qué es un IDE, pasemos a Netbeans. Es un entorno de desarrollo integrado libre, hecho principalmente para el lenguaje de programación Java. Existe además un número importante de módulos para extenderlo. NetBeans IDE es un producto libre y gratuito sin restricciones de uso.



NetBeans es un proyecto de código abierto de gran éxito con una gran base de usuarios, una comunidad en constante crecimiento. Sun Microsystems fundó el proyecto de código abierto NetBeans en junio de 2000 y continúa siendo el patrocinador principal de los proyectos. Actualmente Sun Microsystems es administrado por Oracle Corporation.

La plataforma NetBeans permite que las aplicaciones sean desarrolladas a partir de un conjunto de componentes de software llamados módulos. Un módulo es un archivo Java que contiene clases de java escritas para interactuar con las API de NetBeans y un

archivo especial (manifest file) que lo identifica como módulo. Las aplicaciones construidas a partir de módulos pueden ser extendidas agregándole nuevos módulos. Debido a que los módulos pueden ser desarrollados independientemente, las aplicaciones basadas en la plataforma NetBeans pueden ser extendidas fácilmente por otros desarrolladores de software.

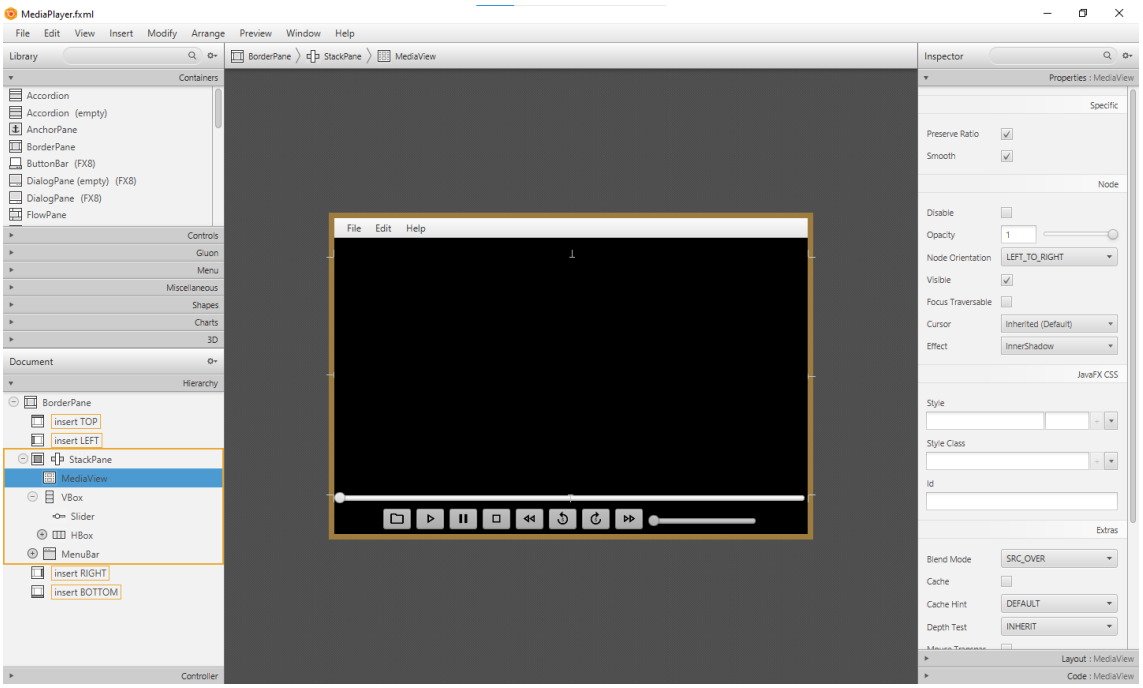
El NetBeans IDE permite el desarrollo de todos los tipos de aplicación Java (J2SE, web, EJB y aplicaciones móviles). Entre sus características se encuentra un sistema de proyectos basado en Ant, control de versiones y refactoring.

Es un framework que simplifica el desarrollo de aplicaciones para Java Swing. El paquete de NetBeans IDE para Java SE contiene lo que se necesita para empezar a desarrollar plugins y aplicaciones basadas en la plataforma NetBeans; no se requiere un SDK adicional.

Las aplicaciones pueden instalar módulos dinámicamente. Algunas aplicaciones pueden incluir un módulo de actualización para permitir a los usuarios descargar Actualizaciones de firma digital y nuevas características directamente dentro de la aplicación en ejecución. Instalando una actualización o una nueva versión, no obligando a los usuarios a descargar toda la aplicación de nuevo.

Tal y como se menciona, Netbeans es un framework para desarrollar aplicaciones en Java Swing, por lo que para nuestro proyecto necesitamos una herramienta capaz de trabajar de manera simplificada con componentes JavaFX, el cual es JavaFX Scene Builder.

JavaFX Scene Builder



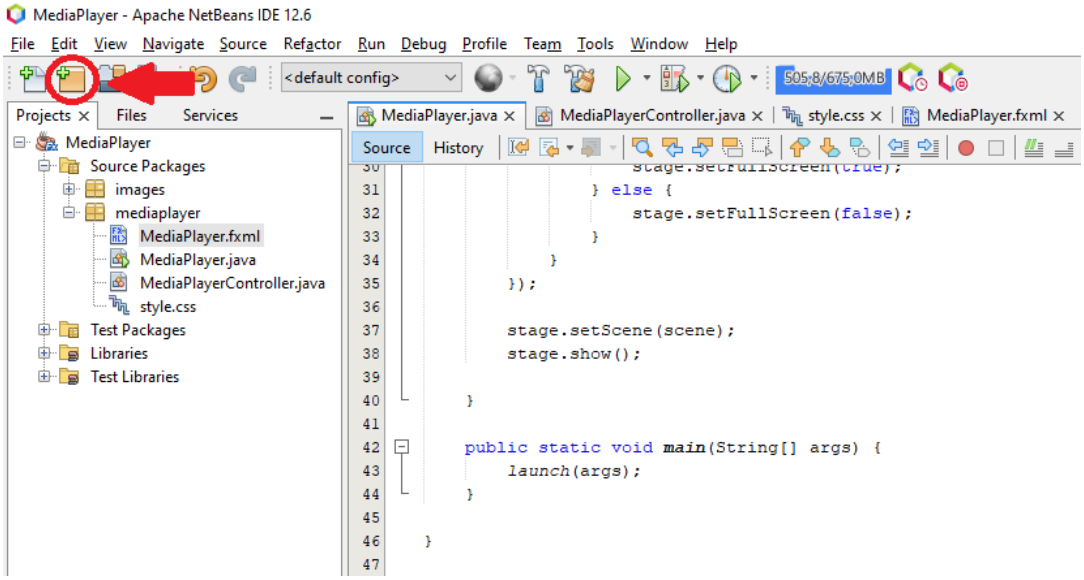
JavaFX Scene Builder es una herramienta de diseño visual que permite a los usuarios diseñar rápidamente interfaces de usuario de aplicaciones JavaFX, sin necesidad de codificación. Los usuarios pueden arrastrar y soltar componentes de interfaz de usuario a un área de trabajo, modificar sus propiedades, aplicar hojas de estilo y el código FXML del diseño que están creando se genera automáticamente en segundo plano. El resultado es un archivo FXML que se puede combinar con un proyecto Java vinculando la interfaz de usuario a la lógica de la aplicación.

La aplicación Scene Builder permite diseñar, mediante un interfaz gráfico, las estructuras de las ventanas de las aplicaciones que queramos desarrollar usando JavaFX. En este artículo podrás conocer los fundamentos básicos para empezar a usar esta herramienta de manera integrada con el entorno de desarrollo NetBeans.

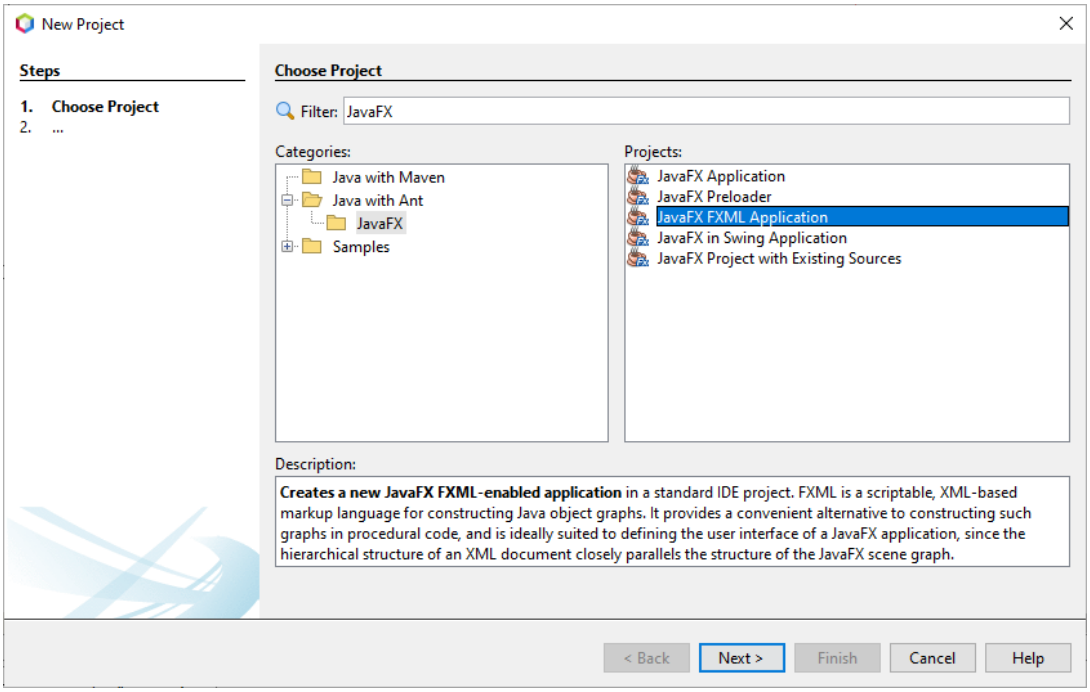
Estructura de directorios y archivos

Para poder explicar la estructura de archivos y directorios de un proyecto JavaFX en Netbeans, primero debemos explicar cómo crearlo.

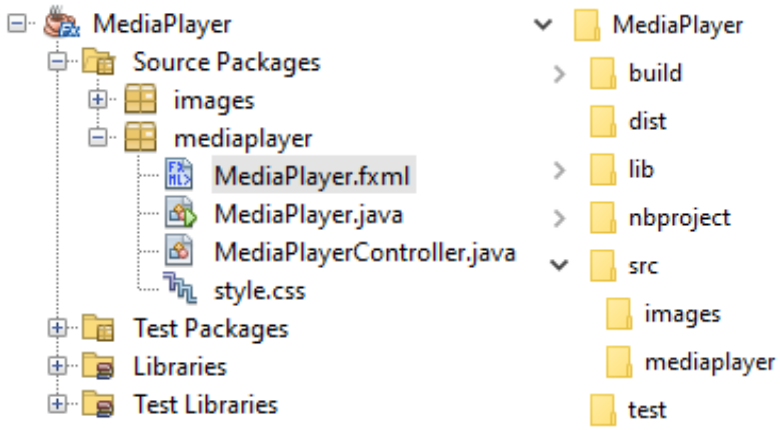
Para crear un proyecto JavaFX en Netbeans debemos hacer clic en el botón de crear nuevo proyecto.



Al hacer clic se nos abrirá la ventana de creación, donde deberemos buscar el tipo de proyecto “JavaFX FXML Application”.



Ahora sí, pasemos a explicar la estructura en sí.



Tenemos 2 imágenes arriba, la de la izquierda es el árbol de directorios y archivos del proyecto visto desde Netbeans, y la de la derecha es el visto desde el propio explorador de Windows.

Como todo proyecto de java hecho en Netbeans, el nuestro tiene una estructura de directorios y archivos “predeterminada”, la cual está explicada a continuación.

Las carpetas más interesantes son las siguientes:

- La carpeta **"build"** contiene las clases compiladas (.class).
- En la carpeta **"dist"** se crea un archivo comprimido con la extensión ".jar" que contiene todos los archivos necesarios para distribuir la aplicación generada y que pueda ser ejecutada.
- La carpeta **"src"** almacena los archivos con el código fuente (.java) que hemos escrito o que forman parte de la aplicación.

Ahora se explicará los archivos de la carpeta “src”, que difieren en número y uso de lo habitual:

- **MediaPlayer.fxml** contiene la descripción de la interfaz de usuario y, por decirlo de manera simple y entendible, es la vista de la aplicación.
- **MediaPlayerController.java** es el archivo java donde se llevan a cabo todas las funciones de las que disponen los controles de nuestra aplicación.
- **MediaPlayer.java** es el modelo que consta de objetos de dominio, definidos en el lado de Java, que se pueden conectar a la vista a través del controlador.
- **style.css** es el archivo donde le aplicamos estilos a nuestro proyecto.

Librerías de Java

java.io

Proporciona la entrada y salida del sistema a través de flujos de datos, serialización y el sistema de archivos.

java.io.File

Una representación abstracta de los nombres de ruta de archivos y directorios.

Las interfaces de usuario y los sistemas operativos utilizan cadenas de texto de rutas dependientes del sistema para nombrar archivos y directorios. Esta clase presenta una visión abstracta de los nombres de ruta jerárquicos independiente del sistema. Un nombre de ruta abstracto tiene dos componentes:

- Una cadena de prefijo opcional dependiente del sistema, como un especificador de unidad de disco, "/" para el directorio raíz de UNIX, o "\\\\" para un nombre de ruta UNC de Microsoft Windows.
- Una secuencia de cero o más nombres de cadena.

El primer nombre de una ruta abstracta puede ser un nombre de directorio o, en el caso de las rutas UNC de Microsoft Windows, un nombre de host. Cada nombre subsiguiente en un nombre de ruta abstracto denota un directorio; el último nombre puede denotar un directorio o un archivo. El nombre abstracto vacío no tiene prefijo y tiene una secuencia de nombres vacía.

La conversión de una cadena de nombres de ruta a o desde un nombre de ruta abstracto depende intrínsecamente del sistema. Cuando un nombre de ruta abstracto se convierte en una cadena de ruta, cada nombre se separa del siguiente por una sola copia del carácter separador por defecto. El carácter separador de nombres por defecto está definido por la propiedad del sistema file.separator, y está disponible en los campos públicos estáticos separator y separatorChar de esta clase. Cuando una cadena de nombres se convierte en un nombre de ruta abstracto, los nombres que contiene pueden estar separados por el carácter separador de nombres por defecto o por cualquier otro carácter separador de nombres soportado por el sistema subyacente.

Un nombre de ruta ya sea abstracto o en forma de cadena, puede ser absoluto o relativo.

Un nombre de ruta absoluto es completo en el sentido de que no se requiere ninguna otra información para localizar el archivo que denota. Un nombre de ruta relativo, por el contrario, debe ser interpretado en términos de información tomada de algún otro nombre de ruta. Por defecto, las clases del paquete java.io siempre resuelven los nombres de ruta relativos contra el directorio actual del usuario. Este directorio es nombrado por la propiedad del sistema user.dir, y es típicamente el directorio en el cual la máquina virtual de Java fue invocada.

El padre de un nombre de ruta abstracto puede obtenerse invocando el método getParent() de esta clase y consiste en el prefijo del nombre de ruta y cada nombre de la secuencia de nombres del nombre de ruta excepto el último. El nombre absoluto de cada directorio es un ancestro de cualquier objeto File con un nombre de ruta abstracto absoluto que comienza con el nombre de ruta absoluto del directorio. Por ejemplo, el directorio indicado por la ruta abstracta "/usr" es un antecesor del directorio indicado por la ruta "/usr/local/bin".

El concepto de prefijo se utiliza para manejar los directorios raíz en las plataformas UNIX, y los especificadores de unidad, los directorios raíz y los nombres de ruta UNC en las plataformas Microsoft Windows, como sigue:

En las plataformas UNIX, el prefijo de una ruta absoluta es siempre "/". Los nombres de ruta relativos no tienen prefijo. El nombre de ruta abstracto que denota el directorio raíz tiene el prefijo "/" y una secuencia de nombres vacía.

En las plataformas Microsoft Windows, el prefijo de un nombre de ruta que contiene un especificador de unidad consiste en la letra de la unidad seguida de ":" y posiblemente seguida de "\" si el nombre de ruta es absoluto. El prefijo de un nombre de ruta UNC es "\\\\"; el nombre del host y el nombre del recurso compartido son los dos primeros nombres de la secuencia de nombres. Un nombre de ruta relativo que no especifica una unidad no tiene prefijo.

Las instancias de esta clase pueden denotar o no un objeto real del sistema de archivos, como un archivo o un directorio. Si denotan un objeto de este tipo, entonces ese objeto reside en una partición. Una partición es una porción de almacenamiento específica del sistema operativo para un sistema de archivos. Un único dispositivo de almacenamiento

(por ejemplo, una unidad de disco física, una memoria flash o un CD-ROM) puede contener varias particiones. El objeto, si lo hay, residirá en la partición nombrada por algún ancestro de la forma absoluta de este nombre de ruta.

Un sistema de archivos puede implementar restricciones a ciertas operaciones en el objeto real del sistema de archivos, como la lectura, la escritura y la ejecución. Estas restricciones se conocen colectivamente como permisos de acceso. El sistema de archivos puede tener varios conjuntos de permisos de acceso sobre un mismo objeto. Por ejemplo, un conjunto puede aplicarse al propietario del objeto, y otro puede aplicarse a todos los demás usuarios. Los permisos de acceso a un objeto pueden hacer que algunos métodos de esta clase fallen.

Las instancias de la clase File son inmutables; es decir, una vez creadas, la ruta abstracta representada por un objeto File nunca cambiará.

java.io.IOException

Señala que se ha producido una excepción de I/O de algún tipo. Esta clase es la clase general de excepciones producidas por operaciones de I/O fallidas o interrumpidas.

java.net

Proporciona las clases para implementar aplicaciones de red.

java.net.URL

La clase URL representa un localizador uniforme de recursos, un puntero a un "recurso" de internet. Un recurso puede ser algo tan simple como un archivo o un directorio, o puede ser una referencia a un objeto más complicado, como una consulta a una base de datos o a un motor de búsqueda.

En general, una URL puede dividirse en varias partes. Considere el siguiente ejemplo:

---

http://www.example.com/docs/resource1.html

La URL anterior indica que el protocolo a utilizar es http (HyperText Transfer Protocol) y que la información reside en una máquina anfitriona llamada www.example.com. La información en esa máquina anfitriona se llama /docs/resource1.html. El significado exacto de este nombre en la máquina anfitriona depende tanto del protocolo como del anfitrión. La información normalmente reside en un archivo, pero podría generarse sobre la marcha. Este componente de la URL se denomina componente de la ruta.

Una URL puede especificar opcionalmente un "puerto", que es el número de puerto al que se realiza la conexión TCP en la máquina anfitriona remota. Si no se especifica el puerto, se utiliza el puerto por defecto para el protocolo. Por ejemplo, el puerto por defecto para http es el 80. Se puede especificar un puerto alternativo como:

---

http://www.example.com:1080/docs/resource1.html

La sintaxis de la URL se define en el RFC 2396: Uniform Resource Identifiers (URI): Generic Syntax, modificado por el RFC 2732: Format for Literal IPv6 Addresses in URLs. El formato de dirección IPv6 literal también admite scope_ids. La sintaxis y el uso de scope_ids se describen aquí.

A una URL se le puede añadir un "fragmento", también conocido como "ref" o "referencia". El fragmento se indica con el carácter de signo agudo "#" seguido de más caracteres. Por ejemplo:

---

http://java.sun.com/index.html#chapter1

Este fragmento no forma técnicamente parte de la URL. Más bien indica que, una vez recuperado el recurso especificado, la aplicación está interesada específicamente en la parte del documento que tiene adjunta la etiqueta chapter1. El significado de una etiqueta es específico del recurso.

Una aplicación también puede especificar una "URL relativa", que contiene sólo la información suficiente para llegar al recurso en relación con otra URL. Las URL relativas se utilizan con frecuencia en las páginas HTML.

La URL relativa no necesita especificar todos los componentes de una URL. Si falta el protocolo, el nombre del host o el número de puerto, el valor se hereda de la URL completamente especificada. El componente de archivo debe ser especificado. El fragmento opcional no se hereda.

La clase URL no codifica ni decodifica por sí misma ningún componente de la URL según el mecanismo de escape definido en RFC2396. Es responsabilidad del llamante codificar cualquier campo que necesite ser escapado antes de llamar a URL, y también decodificar cualquier campo escapado que sea devuelto por URL. Además, como URL no tiene conocimiento del escape de URL, no reconoce la equivalencia entre la forma codificada o decodificada de la misma URL.

Tenga en cuenta que la clase URI realiza el escape de los campos que la componen en determinadas circunstancias. La forma recomendada de gestionar la codificación y descodificación de las URLs es utilizar URI, y convertir entre estas dos clases utilizando toURI() y URI.toURL().

Las clases URLEncoder y URLDecoder también pueden utilizarse, pero sólo para la codificación de formularios HTML, que no es la misma que el esquema de codificación definido en RFC2396.

java.util

Contiene el marco de trabajo de las colecciones, las clases de colección heredadas, el modelo de eventos, las facilidades de fecha y hora, la internacionalización y varias clases de utilidad (un tokenizador de cadenas, un generador de números aleatorios y una matriz de bits).

java.util.Arrays

Esta clase contiene varios métodos para manipular arrays (como ordenar y buscar). Esta clase también contiene una fábrica estática que permite ver los arrays como listas.

Todos los métodos de esta clase lanzan una NullPointerException si la referencia de la matriz especificada es nula, excepto donde se indique.

La documentación de los métodos contenidos en esta clase incluye una breve descripción de las implementaciones. Dichas descripciones deben considerarse notas de implementación, más que partes de la especificación. Los implementadores deben sentirse libres de sustituir otros algoritmos, siempre que se respete la especificación. (Por ejemplo, el algoritmo utilizado por sort(Object[]) no tiene que ser un MergeSort, pero sí tiene que ser estable).

Esta clase es un miembro del Marco de Colecciones de Java.

java.util.List

Una colección ordenada (también conocida como secuencia). El usuario de esta interfaz tiene un control preciso sobre el lugar de la lista donde se inserta cada elemento. El usuario puede acceder a los elementos por su índice entero (posición en la lista), y buscar elementos en la lista.

A diferencia de los conjuntos, las listas suelen permitir elementos duplicados. Más formalmente, las listas suelen permitir pares de elementos e1 y e2 tales que e1.equals(e2), y suelen permitir múltiples elementos nulos si es que permiten elementos nulos. No es inconcebible que alguien quiera implementar una lista que prohíba los duplicados, lanzando excepciones en tiempo de ejecución cuando el usuario intente insertarlos, pero esperamos que este uso sea raro.

La interfaz de la lista impone estipulaciones adicionales, además de las especificadas en la interfaz de la colección, en los contratos de los métodos iterador, add, remove, equals y hashCode. Las declaraciones de otros métodos heredados también se incluyen aquí por comodidad.

La interfaz List proporciona cuatro métodos para el acceso posicional (indexado) a los elementos de la lista. Las listas (al igual que las matrices de Java) se basan en el cero. Tenga en cuenta que estas operaciones pueden ejecutarse en un tiempo proporcional al valor del índice para algunas implementaciones (la clase LinkedList, por ejemplo). Por lo tanto, iterar sobre los elementos de una lista es típicamente preferible a indexar a través de ella si el llamador no conoce la implementación.

La interfaz List proporciona un iterador especial, llamado ListIterator, que permite la inserción y sustitución de elementos y el acceso bidireccional, además de las operaciones normales que proporciona la interfaz Iterator. Se proporciona un método para obtener un iterador de lista que comienza en una posición especificada en la lista.

La interfaz List proporciona dos métodos para buscar un objeto específico. Desde el punto de vista del rendimiento, estos métodos deben utilizarse con precaución. En muchas implementaciones realizarán costosas búsquedas lineales.

La interfaz de la lista proporciona dos métodos para insertar y eliminar eficazmente múltiples elementos en un punto arbitrario de la lista.

Nota: Aunque es permisible que las listas se contengan a sí mismas como elementos, se recomienda extremar la precaución: los métodos equals y hashCode ya no están bien definidos en una lista de este tipo.

Algunas implementaciones de listas tienen restricciones sobre los elementos que pueden contener. Por ejemplo, algunas implementaciones prohíben los elementos nulos, y otras tienen restricciones en los tipos de sus elementos. El intento de añadir un elemento no elegible lanza una excepción no comprobada, normalmente NullPointerException o ClassCastException. El intento de consultar la presencia de un elemento no elegible puede lanzar una excepción, o simplemente puede devolver false; algunas implementaciones mostrarán el primer comportamiento y otras el segundo. En términos más generales, intentar una operación sobre un elemento no elegible cuya finalización no resultaría en la inserción de un elemento no elegible en la lista puede lanzar una excepción o puede tener éxito, a elección de la implementación. Dichas excepciones están marcadas como "opcionales" en la especificación de esta interfaz.

Esta interfaz es un miembro del marco de trabajo de las colecciones de Java.

java.util.ResourceBundle

Los paquetes de recursos contienen objetos específicos de la configuración regional.

Cuando su programa necesita un recurso específico de la configuración regional, una cadena, por ejemplo, su programa puede cargarlo desde el paquete de recursos que es apropiado para la configuración regional del usuario actual. De esta manera, puedes escribir código de programa que es en gran medida independiente de la configuración regional del usuario, aislando la mayor parte, si no toda, la información específica de la configuración regional en los paquetes de recursos.

Esto le permite escribir programas que pueden ser fácilmente localizados, o traducidos, a diferentes idiomas manejar varias configuraciones regionales a la vez ser fácilmente modificados más adelante para admitir aún más configuraciones regionales.

Los paquetes de recursos pertenecen a familias cuyos miembros comparten un nombre base común, pero cuyos nombres también tienen componentes adicionales que identifican sus configuraciones regionales. Por ejemplo, el nombre base de una familia de paquetes de recursos puede ser "MisRecursos". La familia debe tener un paquete de recursos por defecto que simplemente tenga el mismo nombre que su familia - "MisRecursos"- y que se utilizará como paquete de último recurso si no se admite una configuración regional específica. La familia puede entonces proporcionar tantos miembros específicos de una localización como sea necesario, por ejemplo, uno en alemán llamado "MyResources_de".

Cada paquete de recursos de una familia contiene los mismos elementos, pero los elementos han sido traducidos para la configuración regional representada por ese paquete de recursos. Por ejemplo, tanto "MyResources" como "MyResources_de" pueden tener una cadena que se utiliza en un botón para cancelar operaciones. En "MyResources" la cadena puede contener "Cancel" y en "MyResources_de" puede contener "Abbrechen".

Si hay diferentes recursos para diferentes países, puede hacer especializaciones: por ejemplo, "MyResources_de_CH" contiene objetos para el idioma alemán (de) en Suiza (CH). Si quiere modificar sólo algunos de los recursos de la especialización, puede hacerlo.

Cuando su programa necesita un objeto específico de la localidad, carga la clase ResourceBundle utilizando el método getBundle:

```
ResourceBundle myResources =
ResourceBundle.getBundle("MyResources", currentLocale);
```

Los paquetes de recursos contienen pares clave/valor. Las claves identifican de forma exclusiva un objeto específico de la localidad en el paquete.

Las claves son siempre cadenas. En este ejemplo, las claves son "OkKey" y "CancelKey". En el ejemplo anterior, los valores también son cadenas - "Aceptar" y "Cancelar"- pero no tienen por qué serlo. Los valores pueden ser cualquier tipo de objeto.

Usted recupera un objeto del paquete de recursos utilizando el método getter apropiado. Dado que "OkKey" y "CancelKey" son cadenas de caracteres, se utilizará getString para recuperarlas:

```
button1 = new Button(myResources.getString("OkKey"));

button2 = new Button(myResources.getString("CancelKey"));
```

Todos los métodos getter requieren la clave como argumento y devuelven el objeto si lo encuentran. Si no se encuentra el objeto, el método getter lanza una MissingResourceException.

Además de getString, ResourceBundle también proporciona un método para obtener matrices de cadenas, getStringArray, así como un método genérico getObject para cualquier otro tipo de objeto. Al utilizar getObject, tendrás que convertir el resultado al tipo apropiado. Por ejemplo

```
int[] myIntegers = (int[]) myResources.getObject("intList");
```

La plataforma Java ofrece dos subclases de ResourceBundle, ListResourceBundle y PropertyResourceBundle, que proporcionan una forma bastante sencilla de crear recursos. Como ha visto brevemente en un ejemplo anterior, ListResourceBundle gestiona su recurso como una lista de pares clave/valor. PropertyResourceBundle utiliza un archivo de propiedades para gestionar sus recursos.

Si ListResourceBundle o PropertyResourceBundle no se ajustan a sus necesidades, puede escribir su propia subclase de ResourceBundle. Sus subclases deben anular dos métodos: handleGetObject y getKeys().

ResourceBundle.Control

La clase ResourceBundle.Control proporciona la información necesaria para realizar el proceso de carga de paquetes mediante los métodos de fábrica getBundle que toman una instancia de ResourceBundle.Control. Puede implementar su propia subclase para habilitar formatos de paquetes de recursos no estándar, cambiar la estrategia de búsqueda o definir parámetros de almacenamiento en caché. Consulte las descripciones de la clase y del método de fábrica getBundle para más detalles.

Gestión de la caché

Las instancias de paquetes de recursos creadas por los métodos de fábrica getBundle se almacenan en caché por defecto, y los métodos de fábrica devuelven la misma instancia de paquete de recursos varias veces si se ha almacenado en caché. Los clientes de getBundle pueden borrar la caché, gestionar el tiempo de vida de las instancias de paquetes de recursos almacenadas en caché utilizando valores de tiempo de vida o especificar que no se almacenen en caché las instancias de paquetes de recursos. Consulte las descripciones del método de fábrica getBundle, clearCache, ResourceBundle.Control.getTimeToLive y ResourceBundle.Control.needsReload para obtener más detalles.

javafx.application

Proporciona las clases del ciclo de vida de la aplicación.

javafx.application.Application

Clase de aplicación de la que se extienden las aplicaciones JavaFX.

Ciclo de vida

El punto de entrada de las aplicaciones JavaFX es la clase Application. El tiempo de ejecución de JavaFX hace lo siguiente, en orden, cada vez que se lanza una aplicación:

1. Construye una instancia de la clase Application especificada.
2. Llama al método init().
3. Llama al método start(javafx.stage.Stage).
4. Espera a que la aplicación termine, lo que ocurre cuando ocurre cualquiera de los siguientes casos:
    a. la aplicación llama a Platform.exit().
    b. la última ventana se ha cerrado y el atributo implicitExit de Platform es verdadero.
5. Llama al método stop().

Hay que tener en cuenta que el método start es abstracto y debe ser sobrescrito. Los métodos init y stop tienen implementaciones concretas que no hacen nada.

Llamar a Platform.exit() es la forma preferida de terminar explícitamente una aplicación JavaFX. Llamar directamente a System.exit(int) es una alternativa aceptable, pero no permite que se ejecute el método Application.stop().

Una Aplicación JavaFX no debe intentar utilizar JavaFX después de que el kit de herramientas FX haya terminado o desde un ShutdownHook, es decir, después de que el método stop() regrese o se llame a System.exit(int).

Parámetros

Los parámetros de la aplicación están disponibles llamando al método getParameters() desde el método init(), o en cualquier momento después de que el método init haya sido llamado.

Threading

JavaFX crea un hilo de aplicación para ejecutar el método de inicio de la aplicación, procesar los eventos de entrada y ejecutar las líneas de tiempo de la animación. La creación de los objetos JavaFX Scene y Stage, así como la modificación de las operaciones del gráfico de la escena a los objetos vivos (aquellos objetos que ya están adjuntos a una escena) deben realizarse en el hilo de la aplicación JavaFX.

El lanzador Java carga e inicializa la clase de aplicación especificada en el hilo de aplicación JavaFX. Si no hay un método principal en la clase Application, o si el método principal llama a Application.launch(), entonces se construye una instancia de la aplicación en el hilo de aplicación JavaFX.

El método init es llamado en el hilo lanzador, no en el hilo de la aplicación JavaFX. Esto significa que una aplicación no debe construir una escena o un escenario en el método init. Una aplicación puede construir otros objetos JavaFX en el método init.

Todas las excepciones no controladas en el hilo de la aplicación JavaFX que se produzcan durante el envío de eventos, la ejecución de líneas de tiempo de animación o cualquier otro código, se envían al controlador de excepciones no detectadas del hilo.

javafx.beans

El paquete javafx.beans contiene las interfaces que definen la forma más genérica de observabilidad.

javafx.beans.Observable

Un Observable es una entidad que envuelve el contenido y permite observar el contenido para las invalidaciones.

Una implementación de Observable puede soportar la evaluación perezosa, lo que significa que el contenido no se vuelve a calcular inmediatamente después de los cambios, sino perezosamente la próxima vez que se solicite. Todos los enlaces y propiedades de esta biblioteca soportan la evaluación perezosa.

Las implementaciones de esta clase deberían esforzarse por generar el menor número de eventos posible para evitar perder demasiado tiempo en los manejadores de eventos. Las implementaciones de esta biblioteca se marcan como no válidas cuando se produce el primer evento de invalidación. No generan más eventos de invalidación hasta que su valor se recomponga y vuelva a ser válido.

javafx.beans.binding

Características de las fijaciones.

javafx.beans.binding.Bindings

Bindings es una clase de ayuda con un montón de funciones de utilidad para crear enlaces simples.

Normalmente hay dos posibilidades para definir la misma operación: la API de Fluent y los métodos de fábrica de esta clase. Esto permite al desarrollador definir expresiones complejas de la manera más fácil de entender.

La principal diferencia entre utilizar la API de Fluent y los métodos de fábrica de esta clase es que la API de Fluent requiere que al menos uno de los operandos sea una Expresión (véase javafx.beans.binding). (Cada Expression contiene un método estático que genera una Expression a partir de un ObservableValue).

Además, si ha observado con atención, habrá notado que el tipo de retorno de la API Fluent es diferente en los ejemplos anteriores. En muchos casos la API de Fluent permite ser más específico sobre el tipo devuelto (ver NumberExpression para más detalles sobre el casting implícito.

javafx.beans.property

El paquete javafx.beans.property define propiedades de sólo lectura y propiedades de escritura, además de una serie de implementaciones.

javafx.beans.property.DoubleProperty

Esta clase define una Propiedad que envuelve un valor doble.

El valor de una DoubleProperty puede obtenerse y establecerse con ObservableDoubleValue.get(), DoubleExpression.getValue(), WritableDoubleValue.set(double), y setValue(Number).

Una propiedad se puede vincular y desvincular unidireccionalmente con Property.bind(ObservableValue) y Property.unbind(). Las vinculaciones bidireccionales se pueden crear y eliminar con bindBidirectional(Property) y unbindBidirectional(Property).

El contexto de una DoubleProperty puede leerse con ReadOnlyProperty.getBean() y ReadOnlyProperty.getName().

Nota: si se establece o se vincula esta propiedad a un valor nulo, se establecerá la propiedad como "0,0". Véase setValue(java.lang.Number).

javafx.beans.value

El paquete javafx.beans.value contiene las dos interfaces fundamentales ObservableValue y WritableValue y todas sus subinterfaces.

javafx.beans.value.ObservableValue

Un ObservableValue es una entidad que envuelve un valor y permite observar el valor para los cambios. En general, esta interfaz no debe implementarse directamente, sino una de sus subinterfaces ("ObservableBooleanValue", etc.).

El valor del ObservableValue puede solicitarse con getValue().

Una implementación de ObservableValue puede soportar la evaluación perezosa, lo que significa que el valor no se vuelve a calcular inmediatamente después de los cambios, sino perezosamente la próxima vez que se solicite el valor. Todos los enlaces y propiedades de esta biblioteca soportan la evaluación perezosa.

Un ObservableValue genera dos tipos de eventos: eventos de cambio y eventos de invalidación. Un evento de cambio indica que el valor ha cambiado. Un evento de invalidación se genera si el valor actual ya no es válido. Esta distinción es importante si el ObservableValue admite la evaluación perezosa, porque para un valor evaluado perezosamente no se sabe si un valor inválido ha cambiado realmente hasta que se vuelve a calcular. Por esta razón, la generación de eventos de cambio requiere una evaluación ansiosa, mientras que los eventos de invalidación pueden generarse para implementaciones ansiosas y perezosas.

Las implementaciones de esta clase deberían esforzarse por generar el menor número de eventos posible para evitar perder demasiado tiempo en los manejadores de eventos. Las implementaciones de esta biblioteca se marcan como inválidas cuando se produce el primer evento de invalidación. No generan más eventos de invalidación hasta que su valor se recomponga y vuelva a ser válido.

Se pueden adjuntar dos tipos de listeners a un ObservableValue: InvalidationListener para escuchar los eventos de invalidación y ChangeListener para escuchar los eventos de cambio.

Nota importante: adjuntar un ChangeListener hace que se realice un cómputo ansioso incluso si la implementación del ObservableValue admite la evaluación perezosa.

javafx.event

Proporciona un marco básico para los eventos FX, su entrega y manejo.

javafx.event.ActionEvent

Un evento que representa algún tipo de acción. Este tipo de evento se utiliza ampliamente para representar una variedad de cosas, como cuando un botón se ha disparado, cuando un KeyFrame ha terminado, y otros usos similares.

javafx.fxml

Contiene clases para cargar una jerarquía de objetos desde el marcado.

javafx.fxml.FXML

Anotación que etiqueta una clase o miembro como accesible al marcado.

javafx.fxml.FXMLLoader

Carga una jerarquía de objetos desde un documento XML.

javafx.fxml.Initializable

NOTA: Esta interfaz ha sido sustituida por la inyección automática de propiedades de ubicación y recursos en el controlador. FXMLLoader ahora llamará automáticamente a cualquier método initialize() sin carga adecuadamente anotado definido por el controlador. Se recomienda utilizar el método de inyección siempre que sea posible.

La clase es llamada para inicializar un controlador después de que su elemento raíz haya sido completamente procesado.

javafx.geometry

Proporciona el conjunto de clases 2D para definir y realizar operaciones sobre objetos relacionados con la geometría bidimensional.

javafx.geometry.Insets

Un conjunto de desplazamientos interiores para los 4 lados de un área rectangular.

javafx.scene

Proporciona el conjunto de clases base para la API JavaFX Scene Graph.

javafx.scene.Parent

La clase base para todos los nodos que tienen hijos en el gráfico de escena.

Esta clase maneja todas las operaciones jerárquicas del grafo de escena, incluyendo la adición/eliminación de nodos hijos, la marcación de ramas sucias para el trazado y el renderizado, el picking, los cálculos de límites y la ejecución del pase de trazado en cada pulso.

Hay dos subclases concretas directas de Parent:

- Efectos de grupo y transformaciones que se aplican a una colección de nodos hijos.
- Clase Region para nodos que pueden ser estilizados con CSS y layout children.

javafx.scene.Scene

La clase JavaFX Scene es el contenedor de todo el contenido de un gráfico de escena. El fondo de la escena se rellena según lo especificado por la propiedad fill.

La aplicación debe especificar el nodo raíz para el gráfico de la escena estableciendo la propiedad root. Si se utiliza un grupo como raíz, el contenido del gráfico de escena será recortado por el ancho y el alto de la escena y los cambios en el tamaño de la escena (si el usuario redimensiona el escenario) no alterarán el diseño del gráfico de escena. Si se establece un nodo redimensionable (región de diseño o control) como raíz, el tamaño de la raíz seguirá el tamaño de la escena, haciendo que el contenido se retransmita según sea necesario.

El tamaño de la escena puede ser inicializado por la aplicación durante la construcción. Si no se especifica ningún tamaño, la escena calculará automáticamente su tamaño inicial basándose en el tamaño preferido de su contenido. Si sólo se especifica una dimensión, la otra dimensión se calcula utilizando la dimensión especificada, respetando el sesgo del contenido de una raíz.

Una aplicación puede solicitar el soporte del buffer de profundidad o el soporte del anti-aliasing de la escena en la creación de una Escena. Una escena con sólo formas 2D y sin ninguna transformación 3D no necesita un búfer de profundidad ni soporte de antialiasing de escena. Una escena que contenga formas 3D o formas 2D con transformaciones 3D puede utilizar el soporte de la memoria de profundidad para un correcto renderizado clasificado por profundidad; para evitar la lucha por la profundidad (también conocida como lucha por la Z), desactive la prueba de profundidad en las formas 2D que no tengan transformaciones 3D. Consulte depthTest para obtener más información. Una escena con formas 3D puede activar el antialiasing de la escena para mejorar su calidad de renderizado.

Los indicadores depthBuffer y antiAliasing son características condicionales. Con los respectivos valores por defecto de: false y SceneAntialiasing.DISABLED. Ver ConditionalFeature.SCENE3D para más información.

Se añadirá un faro por defecto a una escena que contenga uno o más nodos Shape3D, pero ningún nodo de luz. Esta fuente de luz es un PointLight Color.WHITE colocado en la posición de la cámara.

Los objetos de la escena deben ser construidos y modificados en el hilo de aplicación de JavaFX.

Javafx.scene.control

Los controles de interfaz de usuario de JavaFX (controles UI o simplemente controles) son nodos especializados en el Scenegraph de JavaFX, especialmente adecuados para su reutilización en muchos contextos de aplicación diferentes.

javafx.scene.control.Button

Un simple control de botón. El control de botón puede contener texto y/o un gráfico. Un control de botón tiene tres modos diferentes

- Normal: Un botón normal.
- Por defecto: Un botón por defecto es el que recibe una pulsación del teclado VK_ENTER, si ningún otro nodo de la escena lo consume.
- Cancelar: Un Botón de cancelación es el botón que recibe una pulsación del teclado VK_ESC, si ningún otro nodo de la escena lo consume.

Cuando un botón es presionado y liberado se envía un ActionEvent. Tu aplicación puede realizar alguna acción basada en este evento implementando un EventHandler para procesar el ActionEvent. Los botones también pueden responder a los eventos del ratón implementando un EventHandler para procesar el MouseEvent

MnemonicParsing está activado por defecto para Button.

javafx.scene.control.Hyperlink

Una etiqueta tipo HTML que puede ser un gráfico y/o texto que responde a los rollovers y clics. Cuando un hipervínculo es pulsado/presionado isVisited() se convierte en verdadero. Un hipervínculo se comporta como un botón. Cuando un hipervínculo es presionado y liberado se envía un ActionEvent, y su aplicación puede realizar alguna acción basada en este evento.

javafx.scene.control.Menu

Un menú emergente de elementos accionables que se muestra al usuario sólo cuando lo solicita. Cuando un menú es visible, en la mayoría de los casos de uso, el usuario puede seleccionar un elemento del menú antes de que éste vuelva a su estado oculto. Esto significa que el menú es un buen lugar para poner la funcionalidad importante que no necesariamente tiene que ser visible en todo momento para el usuario.

Los menús suelen colocarse en una MenuBar, o como submenú de otro Menú. Si la intención es ofrecer un menú contextual cuando el usuario hace clic con el botón derecho del ratón en un área determinada de su interfaz de usuario, entonces este no es el control que se debe usar. Esto se debe a que cuando el Menú se añade al scenegraph, tiene una representación visual que hará que aparezca en pantalla. En su lugar, se debe utilizar el ContextMenu en esta circunstancia.

Crear un Menú e insertarlo en una Barra de Menú es fácil, como se muestra a continuación:

```
final Menu menu1 = new Menu("File");

MenuBar menuBar = new MenuBar();

menuBar.getMenus().add(menu1);
```

Un Menú es una subclase de MenuItem, lo que significa que se puede insertar en la lista observable de elementos de un Menú, con lo que se crea un submenú:

```
MenuItem menu12 = new MenuItem("Open");

menu1.getItems().add(menu12);
```

El ObservableList de elementos permite insertar cualquier tipo de MenuItem, incluyendo sus subclases Menu, MenuItem, RadioMenuItem, CheckMenuItem, CustomMenuItem y SeparatorMenuItem. Para insertar un nodo arbitrario en un menú, se puede utilizar un CustomMenuItem. Una excepción a esta regla general es que el SeparatorMenuItem puede utilizarse para insertar un separador.

javafx.scene.control.MenuBar

Un control MenuBar tradicionalmente se coloca en la parte superior de la interfaz de usuario, y dentro de él están los menús. Para añadir un menú a una barra de menús, se añade a la lista observable de menús. Por defecto, para cada menú añadido a la barra de menús, se representará como un botón con el valor de texto Menu mostrado.

MenuBar establece focusTraversable a false.

Para crear y rellenar una MenuBar, puede hacer lo que se muestra a continuación. Por favor, consulte la página de la API del Menú para obtener más información sobre cómo configurarlo.

---

final Menu menu1 = new Menu("File");

final Menu menu2 = new Menu("Opciones");

final Menu menu3 = new Menu("Ayuda");

MenuBar menuBar = new MenuBar();

menuBar.getMenus().addAll(menu1, menu2, menu3);

---

javafx.scene.control.MenuItem

MenuItem está pensado para ser utilizado junto con Menu para proporcionar opciones a los usuarios. MenuItem sirve como clase base para la mayor parte de la API de menús de JavaFX. Tiene una propiedad de texto de visualización, así como un nodo gráfico opcional que se puede establecer en él. La propiedad accelerator permite acceder a la acción asociada con una sola pulsación.

Además, al igual que con el control Button, utilizando el método: setOnAction(javafx.event.EventHandler<javafx.event.ActionEvent>), puede hacer que una instancia de MenuItem realice cualquier acción que desee.

Nota: Aunque se puede insertar cualquier tamaño de gráfico en un MenuItem, el tamaño más utilizado en la mayoría de las aplicaciones es de 16x16 píxeles. Esta es la dimensión gráfica recomendada si se utiliza el estilo por defecto proporcionado por JavaFX.

javafx.scene.control.Separator

Una línea de separación horizontal o vertical. El aspecto visual de este separador puede controlarse mediante CSS. Un separador horizontal ocupa todo el espacio horizontal asignado (menos el relleno), y un separador vertical ocupa todo el espacio vertical asignado (menos el relleno). Las propiedades halignment y valignment determinan cómo se posiciona el separador en la otra dimensión, por ejemplo, cómo se posiciona verticalmente un separador horizontal dentro de su espacio asignado.

El separador es horizontal (es decir, isVertical() == false) por defecto.

La clase de estilo para este control es "separador".

El separador proporciona dos pseudoclases "horizontal" y "vertical" que se excluyen mutuamente. La pseudo-clase "horizontal" se aplica si el separador es horizontal, y la pseudo-clase "vertical" se aplica si el separador es vertical.

El separador establece focusTraversable como falso.

javafx.scene.control.Slider

El control deslizante se utiliza para mostrar un rango continuo o discreto de opciones numéricas válidas y permite al usuario interactuar con el control. Normalmente se representa visualmente como si tuviera una "pista" y un "mando" o "pulgar" que se arrastra dentro de la pista. El control deslizante puede mostrar opcionalmente marcas de verificación y etiquetas que indican los diferentes valores de posición del control deslizante.

Las tres variables fundamentales del deslizador son el mínimo, el máximo y el valor. El valor debe ser siempre un número dentro del rango definido por min y max. min debe ser siempre menor o igual que max (aunque un deslizador cuyo min y max son iguales es un caso degenerado que no tiene sentido). min por defecto es 0, mientras que max por defecto es 100.

javafx.scene.effect

Proporciona el conjunto de clases para adjuntar efectos de filtro gráfico a los nodos de la escena gráfica de JavaFX.

javafx.scene.effect.InnerShadow

Un efecto de alto nivel que renderiza una sombra dentro de los bordes del contenido dado con el color, radio y desplazamiento especificados.

javafx.scene.input

Proporciona el conjunto de clases para el manejo de eventos de entrada de ratón y teclado.

javafx.scene.input.KeyCode

Conjunto de códigos de teclas para los objetos KeyEvent.

javafx.scene.input.KeyEvent

Evento que indica que se ha pulsado una tecla en un Nodo.

Este evento se genera cuando se pulsa, suelta o escribe una tecla. Dependiendo del tipo de evento se pasa a la función onKeyPressed, onKeyTyped o onKeyReleased.

Los eventos "KeyTyped" son de mayor nivel y generalmente no dependen de la plataforma o de la disposición del teclado. Se generan cuando se introduce un carácter Unicode, y son la forma preferida de conocer la entrada de caracteres. En el caso más sencillo, un evento "key typed" se produce al pulsar una sola tecla (por ejemplo, "a"). Sin embargo, a menudo los caracteres son producidos por series de pulsaciones de teclas (por ejemplo, SHIFT + 'a'), y el mapeo de eventos de teclas pulsadas a eventos de teclas tecleadas puede ser de muchos a uno o de muchos a muchos. Normalmente no es necesario soltar una tecla para generar un evento de teclado, pero hay algunos casos en los que el evento de teclado no se genera hasta que se suelta una tecla (por ejemplo, al introducir secuencias ASCII mediante el método Alt-Numpad en Windows). No se

genera ningún evento de tecla tecleada para las teclas que no generan caracteres Unicode (por ejemplo, teclas de acción, teclas modificadoras, etc.).

La variable char siempre contiene un carácter Unicode válido o CHAR_UNDEFINED. La entrada de caracteres se reporta mediante eventos de teclas tecleadas; los eventos de teclas pulsadas y teclas liberadas no están necesariamente asociados a la entrada de caracteres. Por lo tanto, se garantiza que la variable char sólo tiene sentido para los eventos de pulsación de teclas.

Para los eventos de teclas pulsadas y liberadas, la variable code contiene el código de la tecla del evento. Para los eventos de teclado, la variable code siempre contiene KeyCode.UNDEFINED.

Los eventos "tecla pulsada" y "tecla liberada" son de nivel inferior y dependen de la plataforma y de la disposición del teclado. Se generan cada vez que se pulsa o suelta una tecla, y son la única forma de conocer las teclas que no generan entrada de caracteres (por ejemplo, teclas de acción, teclas modificadoras, etc.). La tecla que se pulsa o suelta se indica mediante la variable code, que contiene un código de tecla virtual.

javafx.scene.input.MouseEvent

Cuando se produce un evento del ratón, se escoge el nodo más alto bajo el cursor y se le entrega el evento a través de las fases de captura y burbujeo descritas en EventDispatcher.

La ubicación del ratón (puntero) está disponible en relación con varios sistemas de coordenadas: x,y - en relación con el origen del nodo MouseEvent, sceneX,sceneY - en relación con el origen de la escena que contiene el nodo, screenX,screenY - en relación con el origen de la pantalla que contiene el puntero del ratón.

Gestos de arrastre

Hay tres tipos de gestos de arrastre. Todos ellos son iniciados por un evento de pulsación del ratón y terminados como resultado de un evento de liberación del ratón, el nodo fuente decide qué gesto tendrá lugar.

El gesto simple de pulsar-arrastrar-soltar es el predeterminado. Es el más utilizado para permitir cambiar el tamaño de una forma, arrastrarla, etc. Todo el gesto de presionar-arrastrar-liberar se entrega a un nodo. Cuando se pulsa el botón del ratón, se escoge el nodo más alto y todos los eventos posteriores del ratón se entregan al mismo nodo hasta que se suelta el botón. Si se genera un evento de clic del ratón a partir de estos eventos, se sigue entregando al mismo nodo.

Durante un simple gesto de pulsar-arrastrar-soltar, los otros nodos no están involucrados y no reciben ningún evento. Si estos nodos necesitan estar involucrados en el gesto, el gesto completo de presionar-arrastrar-liberar tiene que ser activado. Este gesto se utiliza mejor para conectar nodos mediante "cables", arrastrar nodos a otros nodos, etc. Este tipo de gesto se describe con más detalle en MouseDragEvent que contiene los eventos entregados a los objetivos del gesto.

El tercer tipo de gesto es el gesto de arrastrar y soltar soportado por la plataforma. Sirve mejor para transferir datos y funciona también entre aplicaciones (no necesariamente FX). Este tipo de gesto se describe con más detalle en DragEvent.

En un breve resumen, el gesto simple de pulsar-arrastrar-soltar se activa automáticamente cuando se pulsa un botón del ratón y entrega todos los MouseEvents a la fuente del gesto. Cuando empiezas a arrastrar, eventualmente llega el evento DRAG_DETECTED. En su manejador puedes iniciar el gesto completo de arrastrar y soltar llamando al método startFullDrag en un nodo o escena - los MouseDragEvents comienzan a ser entregados a los objetivos del gesto, o puedes iniciar el gesto de arrastrar y soltar llamando al método startDragAndDrop en un nodo o escena - el sistema cambia al modo de arrastrar y soltar y los DragEvents comienzan a ser entregados en lugar de los MouseEvents. Si no llamas a ninguno de esos métodos, el simple gesto de pulsar-arrastrar-soltar continúa.

Ten en cuenta que arrastrar un dedo sobre la pantalla táctil produce eventos de arrastre del ratón, pero también eventos de gesto de desplazamiento. Si esto significa un conflicto en una aplicación (la acción de arrastre físico es manejada por dos manejadores diferentes), el método isSynthesized() puede ser usado para detectar el problema y hacer que los manejadores de arrastre se comporten en consecuencia.

Manejo de la entrada/salida del ratón

Cuando el ratón entra en un nodo, el nodo recibe el evento MOUSE_ENTERED, cuando sale, recibe el evento MOUSE_EXITED. Estos eventos se entregan sólo al nodo que ha entrado/salido y aparentemente no pasan por las fases de captura/burbujeo. Este es el caso de uso más común.

Cuando se desea capturar o burbujear, hay eventos MOUSE_ENTERED_TARGET/MOUSE_EXITED_TARGET. Estos eventos pasan por las fases de captura/burbujeo normalmente. Esto significa que el padre puede recibir el evento MOUSE_ENTERED_TARGET cuando el ratón ha entrado en el propio padre o en alguno de sus hijos. Para distinguir entre estos dos casos se puede comprobar la igualdad del evento objetivo con el nodo.

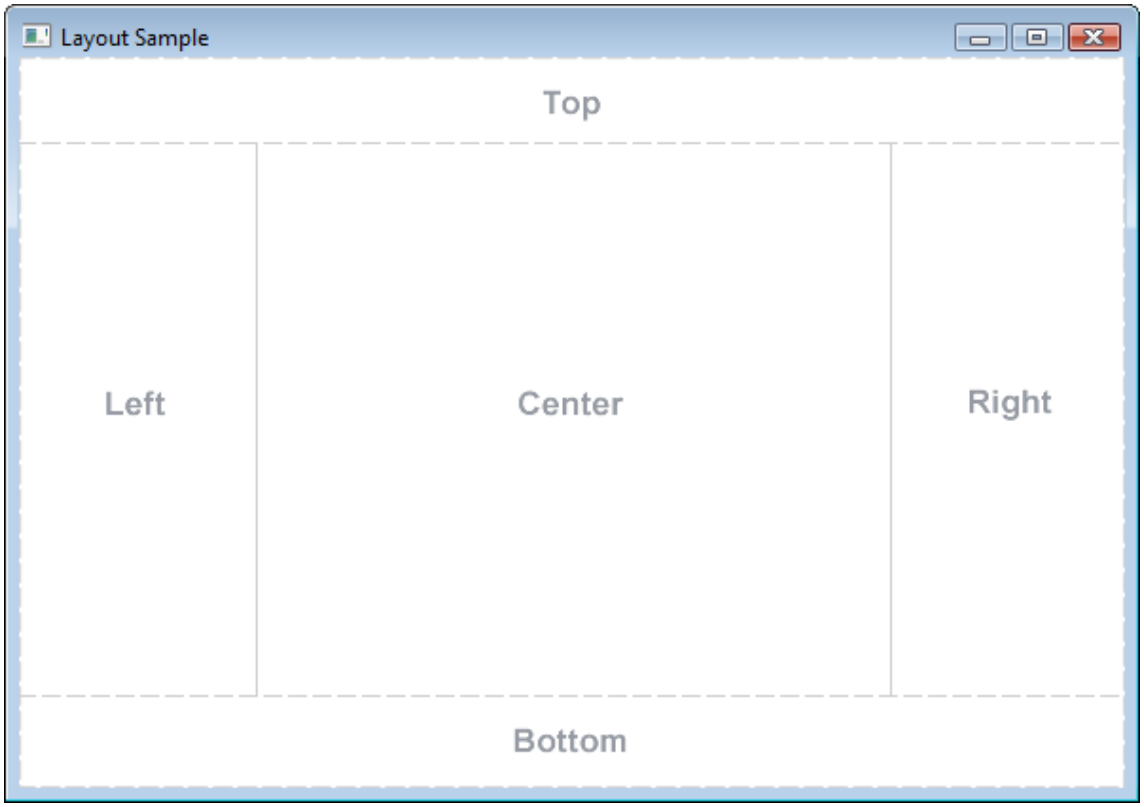
Estos dos tipos están estrechamente relacionados: MOUSE_ENTERED/MOUSE_EXITED son subtipos de MOUSE_ENTERED_TARGET/MOUSE_EXITED_TARGET. Durante la fase de captura, MOUSE_ENTERED_TARGET se entrega a los padres. Cuando el evento se entrega al objetivo del evento (el nodo que realmente ha sido introducido), su tipo se cambia a MOUSE_ENTERED. Entonces el tipo se cambia de nuevo a MOUSE_ENTERED_TARGET para la fase de burbujeo. Sigue siendo un solo evento que cambia de tipo, por lo que, si se filtra o se consume, afecta a ambas variantes del evento. Gracias a la relación de subtipo, un manejador de eventos MOUSE_ENTERED_TARGET recibirá el evento MOUSE_ENTERED en el objetivo.

javafx.scene.layout

Proporciona clases para apoyar el diseño de la interfaz de usuario.

javafx.scene.layout.BorderPane

BorderPane dispone los hijos en las posiciones superior, izquierda, derecha, inferior y central.



Los hijos superiores e inferiores serán redimensionados a sus alturas preferidas y extenderán el ancho del panel del borde. Los hijos izquierdo y derecho serán redimensionados a sus anchos preferidos y extenderán la longitud entre los nodos superior e inferior. Y el nodo central se redimensionará para llenar el espacio disponible en el centro. Cualquiera de las posiciones puede ser nula.

Los bordes pueden ser estilizados con fondos y bordes usando CSS. Vea la superclase Region para más detalles.

BorderPane respeta los tamaños mínimo, preferido y máximo de sus hijos. Si el rango de redimensionamiento del hijo impide que sea redimensionado para encajar en su posición, se alineará en relación con el espacio utilizando una alineación por defecto como la siguiente:

- top: Pos.TOP_LEFT
- inferior: Pos.BOTTOM_LEFT
- izquierda: Pos.TOP_LEFT
- derecha: Pos.TOP_RIGHT
- centro: Pos.CENTER

BorderPane coloca cada conjunto de hijos en las cinco posiciones independientemente del valor de la propiedad visible del hijo; los hijos no administrados se ignoran.

Alcance redimensionable

El BorderPane se utiliza comúnmente como la raíz de una escena, en cuyo caso su tamaño seguirá el tamaño de la escena. Si el tamaño de la escena o del escenario no ha sido establecido directamente por la aplicación, el tamaño de la escena será inicializado al tamaño preferido del panel de borde. Sin embargo, si un panel de borde tiene un padre que no es la escena, ese padre cambiará el tamaño del panel de borde dentro del rango de cambio de tamaño del panel de borde durante el diseño. Por defecto, el panel de borde calcula este rango basándose en su contenido, como se indica en la tabla siguiente:

	width	**height**
Mínimo	los insertos izquierdo/derecho más la anchura necesaria para mostrar a los hijos de la derecha/izquierda con sus anchuras prefijadas y arriba/abajo/centro con al menos sus anchuras mínimas	los insertos superior/inferior más la altura requerida para mostrar los hijos superiores/inferiores a sus alturas prefijadas y la izquierda/derecha/centro con al menos sus alturas mínimas
Preferido	inserciones izquierda/derecha más la anchura necesaria para mostrar los hijos de arriba/derecha/inferior/izquierda/centro con al menos sus anchuras prefijadas	inserciones superiores/inferiores más la altura necesaria para mostrar a los hijos de arriba/derecha/inferior/izquierda/centro con al menos sus alturas prefijadas
Máximo	Double.MAX_VALUE	Double.MAX_VALUE

La anchura y la altura máximas no delimitadas de un panel de borde son una indicación para el padre de que puede ser redimensionado más allá de su tamaño preferido para llenar cualquier espacio que se le asigne.

BorderPane proporciona propiedades para establecer el rango de tamaño directamente. Estas propiedades tienen como valor predeterminado el valor centinela Region.USE_COMPUTED_SIZE, aunque la aplicación puede establecer otros valores según sea necesario:

```
borderPane.setPrefSize(500,400);
```

Las aplicaciones pueden restaurar los valores calculados estableciendo estas propiedades de nuevo a Region.USE_COMPUTED_SIZE.

BorderPane no recorta su contenido de forma predeterminada, por lo que es posible que los límites de los hijos se extiendan fuera de sus propios límites si el tamaño mínimo de un hijo impide que se ajuste a su espacio.

Restricciones de diseño opcionales

Una aplicación puede establecer restricciones en los hijos individuales para personalizar el diseño de BorderPane. Para cada restricción, BorderPane proporciona un método estático para establecerla en el hijo.

Restricción	Tipo	Descripción
alineación	javafx.geometry.Pos	La alineación del hijo dentro de su área del panel de borde.
margen	javafx.geometry.Insets	Espacio de margen alrededor del exterior del hijo.

javafx.scene.layout.ColumnConstraints

Define las restricciones de diseño opcionales para una columna en un GridPane. Si se añade un objeto ColumnConstraints para una columna en un gridpane, el gridpane utilizará esos valores de restricción cuando calcule el ancho y el diseño de la columna.

Tenga en cuenta que añadir un objeto ColumnConstraints vacío tiene el efecto de no establecer ninguna restricción, dejando que el GridPane calcule el diseño de la columna basándose únicamente en las preferencias y restricciones de tamaño de su contenido.

javafx.scene.layout.GridPane

GridPane dispone sus hijos dentro de una rejilla flexible de filas y columnas. Si se establece un borde y/o un relleno, entonces su contenido se dispondrá dentro de esas inserciones.

Un hijo puede ser colocado en cualquier lugar dentro de la cuadrícula y puede abarcar múltiples filas/columnas. Los hijos pueden superponerse libremente dentro de las filas/columnas y su orden de apilamiento será definido por el orden de la lista de hijos del gridpane (el 0º nodo en la parte trasera, el último nodo en la parte delantera).

El GridPane puede ser estilizado con fondos y bordes usando CSS. Vea la superclase Region para más detalles.

Restricciones de la rejilla

La ubicación de un hijo dentro de la rejilla está definida por sus restricciones de diseño:

Restricción	Tipo	Descripción
columnIndex	integer	columna donde comienza el área de diseño del hijo
rowIndex	integer	fila donde comienza el área de diseño del hijo
columnSpan	integer	el número de columnas que el área de diseño del hijo abarca horizontalmente
rowSpan	integer	el número de filas que el área de diseño del hijo abarca verticalmente

Si los índices de fila/columna no se establecen explícitamente, el hijo se colocará en la primera fila/columna. Si no se establecen los intervalos de fila/columna, se colocarán por defecto en 1. Las restricciones de colocación de un hijo pueden cambiarse dinámicamente y el gridpane se actualizará en consecuencia.

El número total de filas/columnas no necesita ser especificado por adelantado ya que el gridpane expandirá/contraerá automáticamente la rejilla para acomodar el contenido.

Para utilizar el GridPane, una aplicación necesita establecer las restricciones de diseño en los hijos y añadir esos hijos a la instancia del gridpane. Las restricciones se establecen en los hijos utilizando métodos estáticos de la clase GridPane.

Las aplicaciones también pueden utilizar métodos de conveniencia que combinan los pasos de establecer las restricciones y añadir los hijos.

Tamaño de las filas/columnas

Por defecto, las filas y columnas tendrán el tamaño adecuado para su contenido; una columna será lo suficientemente ancha para acomodar al hijo más ancho, una fila lo suficientemente alta para acomodar al hijo más alto. Sin embargo, si una aplicación necesita controlar explícitamente el tamaño de las filas o columnas, puede hacerlo añadiendo objetos RowConstraints y ColumnConstraints para especificar esas métricas.

Por defecto, el gridpane redimensionará las filas/columnas a sus tamaños preferidos (calculados a partir del contenido o fijos), incluso si el gridpane se redimensiona más allá de su tamaño preferido. Si una aplicación necesita que una fila o columna en particular crezca si hay espacio extra, puede establecer su prioridad de crecimiento en el objeto RowConstraints o ColumnConstraints

Nota: Los nodos que abarcan varias filas/columnas también serán dimensionados a los tamaños preferidos. Las filas/columnas afectadas son redimensionadas por la siguiente prioridad: prioridades de crecimiento, última fila. Esto es con respecto a las restricciones de filas/columnas.

Tamaño porcentual

Alternativamente, RowConstraints y ColumnConstraints permiten especificar el tamaño como un porcentaje del espacio disponible del gridpane.

Si se establece un valor de porcentaje en una fila/columna, entonces ese valor tiene prioridad y las restricciones min, pref, max y grow de la fila/columna serán ignoradas.

Tenga en cuenta que si la suma de los valores de widthPercent (o heightPercent) es superior a 100, los valores se tratarán como pesos. Por ejemplo, si a 3 columnas se les asigna un widthPercent de 50, entonces a cada una se le asignará 1/3 de la anchura disponible del gridpane (50/(50+50+50)).

Mezcla de tipos de tamaño

Una aplicación puede mezclar libremente los tipos de tamaño de las filas/columnas (calculado a partir del contenido, fijo o porcentual). A las filas/columnas porcentuales siempre se les asignará el espacio en primer lugar, basándose en su porcentaje del espacio disponible de la rejilla (el tamaño menos los encajes y los huecos). El espacio restante se asignará a las filas/columnas teniendo en cuenta sus tamaños mínimo, preferido y máximo y sus prioridades de crecimiento.

Rango redimensionable

El padre de un gridpane redimensionará el gridpane dentro del rango de redimensionamiento del gridpane durante el diseño. Por defecto, el gridpane calcula este rango basándose en su contenido y en las restricciones de filas/columnas, como se indica en la tabla siguiente.

	width	**height**
Mínimo	los insertos izquierdo/derecho más la suma de la anchura mínima de cada columna	los insertos superior/inferior más la suma de la altura mínima de cada fila
Preferido	los insertos izquierdo/derecho más la suma de la anchura prefijada de cada columna	los insertos superior/inferior más la suma de la altura prefijada de cada fila
Máximo	Double.MAX_VALUE	Double.MAX_VALUE

La anchura y la altura máximas no delimitadas de un gridpane son una indicación al padre de que puede ser redimensionado más allá de su tamaño preferido para llenar cualquier espacio que se le asigne.

GridPane proporciona propiedades para establecer el rango de tamaño directamente. Estas propiedades tienen por defecto el valor centinela USE_COMPUTED_SIZE, pero la aplicación puede establecer otros valores según sea necesario.

Las aplicaciones pueden restaurar los valores calculados estableciendo estas propiedades de nuevo a USE_COMPUTED_SIZE.

GridPane no recorta su contenido por defecto, por lo que es posible que los límites de los hijos se extiendan fuera de sus propios límites si el tamaño mínimo de un hijo impide que se ajuste a su espacio.

Restricciones de diseño opcionales

Una aplicación puede establecer restricciones adicionales en los hijos para personalizar el tamaño y la posición del hijo dentro del área de diseño establecida por sus índices/espacios de filas/columnas:

Restricción	Tipo	Descripción
halignment	javafx.geometry.HPos	La alineación horizontal del hijo dentro de su área de diseño
valignment	javafx.geometry.VPos	La alineación vertical del hijo dentro de su área de diseño
hgrow	javafx.scene.layout.Priority	La prioridad de crecimiento horizontal del hijo
vgrow	javafx.scene.layout.Priority	La prioridad de crecimiento vertical del hijo
margin	javafx.geometry.Insets	Espacio de margen alrededor del exterior del hijo

Por defecto, la alineación de un hijo dentro de su área de diseño está definida por la alineación establecida para la fila y la columna. Si se establece una restricción de alineación individual en un hijo, esa alineación anulará la alineación de la fila/columna sólo para ese hijo. La alineación de otros hijos en la misma fila o columna no se verá afectada.

Las prioridades de crecimiento, por otro lado, sólo pueden aplicarse a filas o columnas enteras. Por lo tanto, si se establece una restricción de prioridad de crecimiento en un solo hijo, se utilizará para calcular la prioridad de crecimiento por defecto de la fila/columna que lo engloba. Si se establece una prioridad de crecimiento directamente en un objeto RowConstraint o ColumnConstraint, anulará el valor calculado a partir del contenido.

javafx.scene.layout.HBox

HBox dispone sus hijos en una sola fila horizontal. Si el hbox tiene un borde y/o un relleno, entonces el contenido se distribuirá dentro de esas inserciones.

Ejemplo de HBox:

HBox hbox = new HBox(8); // spacing = 8

hbox.getChildren().addAll(new Label("Nombre:), new TextBox());

HBox redimensionará los hijos (si son redimensionables) a sus anchos preferidos y utiliza su propiedad fillHeight para determinar si redimensiona sus alturas para llenar su propia altura o mantiene sus alturas preferidas (fillHeight por defecto es true). La alineación del contenido es controlada por la propiedad alignment, que por defecto es Pos.TOP_LEFT.

Si un hbox se redimensiona más allá de su anchura preferida, por defecto mantendrá los hijos a su anchura preferida, dejando el espacio extra sin utilizar. Si una aplicación desea que a uno o más hijos se les asigne ese espacio extra, puede establecer opcionalmente una restricción hgrow en el hijo. Consulte "Restricciones de diseño opcionales" para más detalles.

HBox distribuye cada hijo gestionado independientemente del valor de la propiedad visible del hijo; los hijos no gestionados se ignoran.

Rango redimensionable

El padre de un hbox redimensionará el hbox dentro del rango de redimensionamiento del hbox durante la maquetación. Por defecto, el hbox calcula este rango basándose en su contenido como se indica en la tabla siguiente.

	width	**height**
Mínimo	los insertos izquierdo/derecho más la suma de la anchura mínima de cada hijo más el espacio entre cada hijo	los insumos superiores/inferiores más la mayor de las alturas mínimas de los hijos
Preferido	los insertos izquierdo/derecho más la suma de la anchura del prefijo de cada hijo más el espacio entre cada hijo	los insertos superior/inferior más la mayor de las alturas de los prefijos de los hijos
Máximo	Double.MAX_VALUE	Double.MAX_VALUE

La anchura y la altura máximas no limitadas de un hbox son una indicación al padre de que puede ser redimensionado más allá de su tamaño preferido para llenar cualquier espacio que se le asigne.

HBox proporciona propiedades para establecer el rango de tamaño directamente. Estas propiedades tienen por defecto el valor centinela USE_COMPUTED_SIZE, sin embargo, la aplicación puede establecer otros valores según sea necesario:

```
hbox.setPrefWidth(400);
```

Las aplicaciones pueden restaurar los valores calculados estableciendo estas propiedades de nuevo a USE_COMPUTED_SIZE.

HBox no recorta su contenido por defecto, por lo que es posible que los límites de los hijos se extiendan fuera de sus propios límites si el tamaño mínimo de un hijo impide que quepa dentro de hbox.

Restricciones de diseño opcionales

Una aplicación puede establecer restricciones en hijos individuales para personalizar el diseño de HBox. Para cada restricción, HBox proporciona un método estático para establecerla en el hijo.

Restricción	Tipo	Descripción
hgrow	javafx.scene.layout.Priority	La prioridad de crecimiento horizontal para el hijo
margin	javafx.geometry.Insets	Espacio de margen alrededor del exterior del hijo

Si más de un hijo tiene la misma prioridad de crecimiento, el hbox asignará la misma cantidad de espacio a cada uno. HBox sólo crecerá un hijo hasta su anchura máxima, por lo que si el hijo tiene una anchura máxima distinta de Double.MAX_VALUE, la aplicación puede necesitar anular el máximo para permitir que crezca.

javafx.scene.layout.RowConstraints

Define restricciones opcionales de diseño para una fila en un GridPane. Si se añade un objeto RowConstraints para una fila en un gridpane, el gridpane utilizará esos valores de restricción cuando calcule la altura y el diseño de la fila.

javafx.scene.layout.StackPane

StackPane dispone sus hijos en una pila de atrás hacia adelante.

El orden z de los hijos está definido por el orden de la lista de hijos, siendo el hijo número 0 el inferior y el último el superior. Si se ha establecido un borde y/o un relleno, los hijos se dispondrán dentro de esos márgenes.

La pantalla apilada intentará cambiar el tamaño de cada hijo para llenar su área de contenido. Si el hijo no puede ser dimensionado para llenar el stackpane (ya sea porque no es redimensionable o porque su tamaño máximo lo impide) entonces será alineado dentro del área usando la propiedad alignment, que por defecto es Pos.CENTER.

StackPane presenta cada hijo gestionado independientemente del valor de la propiedad visible del hijo; los hijos no gestionados se ignoran.

StackPane puede ser estilizado con fondos y bordes usando CSS. Consulte la sección Región para obtener más detalles.

Rango redimensionable

El padre de un stackpane redimensionará el stackpane dentro del rango redimensionable del stackpane durante el diseño. Por defecto, el stackpane calcula este rango basándose en su contenido como se indica en la tabla siguiente.

	width	**height**
Mínimo	los insertos izquierdo/derecho más la mayor de las anchuras mínimas de los hijos	los insumos superiores/inferiores más la mayor de las alturas mínimas de los hijos
Preferido	los insertos izquierda/derecha más la mayor de las anchuras prefijadas de los hijos	los insertos superior/inferior más la mayor de las alturas de los prefijos de los hijos
Máximo	Double.MAX_VALUE	Double.MAX_VALUE

La anchura y la altura máximas no limitadas de un stackpane son una indicación al padre de que puede ser redimensionado más allá de su tamaño preferido para llenar cualquier espacio que se le asigne.

StackPane proporciona propiedades para establecer el rango de tamaño directamente. Estas propiedades tienen por defecto el valor centinela USE_COMPUTED_SIZE, sin embargo, la aplicación puede establecer otros valores según sea necesario:

// hay que asegurar que el stackpane nunca se redimensiona más allá de su tamaño preferido

stackpane.setMaxSize(Region.USE_PREF_SIZE, Region.USE_PREF_SIZE);

Las aplicaciones pueden restaurar los valores calculados estableciendo estas propiedades de nuevo a USE_COMPUTED_SIZE.

StackPane no recorta su contenido de forma predeterminada, por lo que es posible que los límites de los hijos se extiendan fuera de sus propios límites si el tamaño mínimo de un hijo impide que se ajuste al stackpane.

Restricciones de diseño opcionales

Una aplicación puede establecer restricciones en hijos individuales para personalizar el diseño de StackPane. Para cada restricción, StackPane proporciona un método estático para establecerla en el hijo.

Restricción	Tipo	Descripción
alineamiento	javafx.geometry.Pos	La alineación del hijo dentro del stackpane
margen	javafx.geometry.Insets	Espacio de margen alrededor del exterior del hijo

javafx.scene.layout.VBox

VBox presenta a sus hijos en una sola columna vertical. Si el vbox tiene un borde y/o un relleno establecido, entonces el contenido se dispondrá dentro de esas inserciones.

VBox redimensionará a los hijos (si son redimensionables) a sus alturas preferidas y utiliza su propiedad fillWidth para determinar si redimensiona sus anchos para llenar su propio ancho o mantiene sus anchos a los preferidos (fillWidth por defecto es true). La alineación del contenido es controlada por la propiedad alignment, que por defecto es Pos.TOP_LEFT.

Si un vbox se redimensiona más allá de su altura preferida, por defecto mantendrá los hijos a sus alturas preferidas, dejando el espacio extra sin utilizar. Si una aplicación desea que a uno o más hijos se les asigne ese espacio extra, puede establecer opcionalmente una restricción vgrow en el hijo. Ver "Restricciones opcionales de diseño" para más detalles.

VBox distribuye cada hijo gestionado independientemente del valor de la propiedad visible del hijo; los hijos no gestionados se ignoran.

Rango redimensionable

El padre de un vbox redimensionará el vbox dentro del rango de redimensionamiento del vbox durante la disposición. Por defecto, el vbox calcula este rango basándose en su contenido como se indica en la tabla siguiente.

	width	**height**
Mínimo	los insertos izquierdo/derecho más la mayor de las anchuras mínimas de los hijos	los insertos superior/inferior más la suma de la altura mínima de cada hijo más el espacio entre cada hijo
Preferido	los insertos izquierda/derecha más la mayor de las anchuras prefijadas de los hijos	los insertos superior/inferior más la suma de la altura del prefijo de cada hijo más el espacio entre cada hijo
Máximo	Double.MAX_VALUE	Double.MAX_VALUE

La anchura y la altura máximas no limitadas de un vbox son una indicación al padre de que puede ser redimensionado más allá de su tamaño preferido para llenar cualquier espacio que se le asigne.

VBox proporciona propiedades para establecer el rango de tamaño directamente. Estas propiedades tienen por defecto el valor centinela USE_COMPUTED_SIZE, sin embargo, la aplicación puede establecer otros valores según sea necesario:

```
vbox.setPrefWidth(400);
```

Las aplicaciones pueden restaurar los valores calculados estableciendo estas propiedades de nuevo a USE_COMPUTED_SIZE.

VBox no recorta su contenido por defecto, por lo que es posible que los límites de los hijos se extiendan fuera de sus propios límites si el tamaño mínimo de un hijo impide que quepa dentro de vbox.

Restricciones de diseño opcionales

Una aplicación puede establecer restricciones en hijos individuales para personalizar el diseño de VBox. Para cada restricción, VBox proporciona un método estático para establecerla en el hijo.

Restricción	Tipo	Descripción
vgrow	javafx.scene.layout.Priority	la prioridad de crecimiento vertical para el hijo
margen	javafx.geometry.Insets	espacio de margen alrededor del exterior del hijo

Si más de un hijo tiene la misma prioridad de crecimiento, entonces vbox asignará cantidades iguales de espacio a cada uno. VBox sólo crecerá un hijo hasta su altura máxima, por lo que si el hijo tiene una altura máxima diferente a Double.MAX_VALUE, la aplicación puede necesitar anular el máximo para permitirle crecer.

Javafx.scene.media

Proporciona el conjunto de clases para integrar el audio y el vídeo en las aplicaciones Java FX.

javafx.scene.media.Media

La clase Media representa un recurso multimedia. Se instancia a partir de la forma de cadena de un URI de origen. La información sobre el medio, como la duración, los metadatos, las pistas y la resolución de vídeo, puede obtenerse a partir de una instancia de Media. La información de los medios se obtiene de forma asíncrona, por lo que no está necesariamente disponible inmediatamente después de la instanciación de la clase. Sin embargo, toda la información debería estar disponible si la instancia ha sido asociada a un MediaPlayer y ese reproductor ha pasado al estado MediaPlayer.Status.READY. Para ser notificado cuando se añaden metadatos o pistas, se pueden registrar observadores con las colecciones devueltas por getMetadata()y getTracks(), respectivamente.

El mismo objeto Media puede ser compartido por varios objetos MediaPlayer. Dicha instancia compartida puede gestionar una única copia de los datos multimedia de origen para que la utilicen todos los reproductores, o puede requerir una copia separada de los datos para cada reproductor. Sin embargo, la elección de la implementación no tendrá ningún efecto en el comportamiento del reproductor a nivel de interfaz.

javafx.scene.media.MediaPlayer

La clase MediaPlayer proporciona los controles para la reproducción de medios. Se utiliza en combinación con las clases Media y MediaView para mostrar y controlar la reproducción de medios. MediaPlayer no contiene ningún elemento visual, por lo que debe ser utilizado con la clase MediaView para ver cualquier pista de vídeo que pueda estar presente.

MediaPlayer proporciona los controles pause(), play(), stop() y seek() así como las propiedades rate y autoPlay que se aplican a todos los tipos de medios. También proporciona las propiedades balance, mute y volumen que controlan las características de la reproducción de audio. Se puede obtener un mayor control sobre la calidad del audio a través del AudioEqualizer asociado al reproductor. Los descriptores de frecuencia de la reproducción de audio pueden observarse registrando un AudioSpectrumListener. La información sobre la posición de la reproducción, la velocidad y el almacenamiento en búfer puede obtenerse de las propiedades currentTime, currentRate y bufferProgressTime, respectivamente. Las notificaciones de los marcadores de medios son recibidas por un manejador de eventos registrado como la propiedad onMarker.

Para los medios de duración finita, la reproducción puede situarse en cualquier punto del tiempo entre 0,0 y la duración del medio. MediaPlayer refina esta definición añadiendo las propiedades startTime y stopTime que, en efecto, definen una fuente de medios virtual con una posición de tiempo limitada a [startTime,stopTime]. La reproducción de los medios comienza en startTime y continúa hasta stopTime. El intervalo definido por estos dos puntos finales se denomina ciclo y su duración es la diferencia entre las horas de inicio y parada. Este ciclo puede configurarse para que se repita un número específico o indefinido de veces. La duración total de la reproducción

de medios es entonces el producto de la duración del ciclo y el número de veces que se reproduce el ciclo. Si se alcanza el tiempo de parada del ciclo y éste debe reproducirse de nuevo, se invoca el manejador de eventos registrado con la propiedad onRepeat. Si se alcanza el tiempo de parada y el ciclo no se va a repetir, entonces se invoca el manejador de eventos registrado con la propiedad onEndOfMedia. Un índice relativo a cero de qué ciclo se está reproduciendo actualmente se mantiene mediante currentCount.

El funcionamiento de un MediaPlayer es inherentemente asíncrono. Un reproductor no está preparado para responder a los comandos de forma casi inmediata hasta que su estado haya pasado a MediaPlayer.Status.READY, lo que en efecto ocurre generalmente cuando se completa el pre-roll de los medios. Sin embargo, algunas peticiones realizadas a un reproductor antes de que su estado sea READY surtirán efecto cuando dicho estado sea introducido. Esto incluye la invocación de play() sin una invocación intermedia de pause() o stop() antes de la transición a READY, así como el establecimiento de cualquiera de las propiedades autoPlay, balance, mute, rate, startTime, stopTime y volume.

La propiedad status puede ser monitorizada para que la aplicación sea consciente de los cambios de estado del reproductor, y las funciones de callback pueden ser registradas a través de propiedades como onReady si se debe realizar una acción cuando se entra en un estado particular. También existen las propiedades error y onError que permiten, respectivamente, monitorizar cuando se produce un error y tomar una acción especificada en respuesta al mismo.

El mismo objeto MediaPlayer puede ser compartido entre múltiples MediaViews. Esto no afectará al propio reproductor. En particular, la configuración de las propiedades de la vista no tendrá ningún efecto sobre la reproducción de los medios.

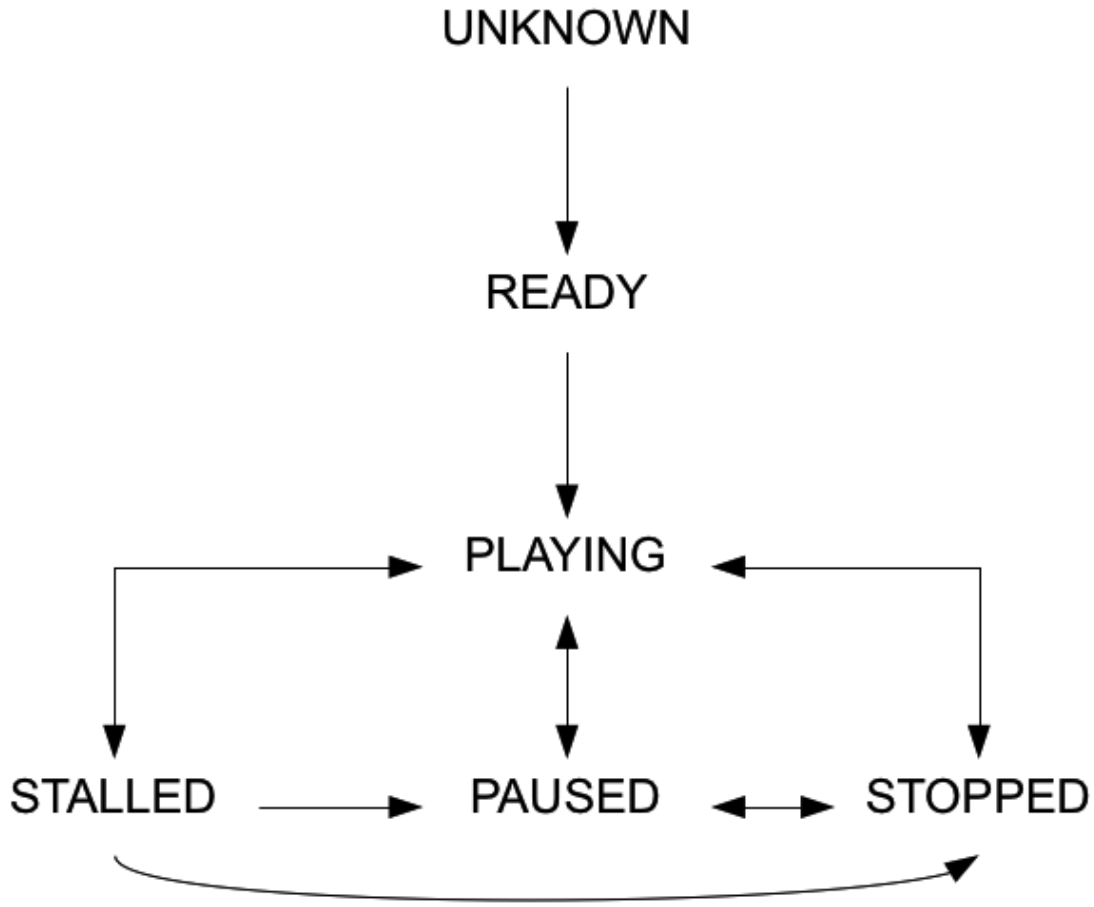
javafx.scene.media.MediaView

Un nodo que proporciona una vista de los medios de comunicación que se reproducen por un MediaPlayer.

El siguiente fragmento de código proporciona un ejemplo sencillo de un método Application.start() que muestra un vídeo.

javafx.scene.media.MediaPlayer.Status

Enumeración que describe los diferentes valores de estado de un MediaPlayer.



javafx.scene.text

Proporciona el conjunto de clases para las fuentes y el nodo de texto renderizable.

javafx.scene.text.Font

La clase Font representa las fuentes, que se utilizan para representar el texto en la pantalla.

El tamaño de una fuente se describe como especificado en puntos, que es una medida del mundo real de aproximadamente 1/72 pulgadas.

Dado que las fuentes se escalan con la transformación de renderizado determinada por los atributos de transformación de un nodo que utiliza la fuente y sus ancestros, el tamaño será en realidad relativo al espacio de coordenadas local del nodo, que debería proporcionar coordenadas similares al tamaño de un punto si no hay transformaciones de escala en el entorno del nodo. Tenga en cuenta que las distancias del mundo real especificadas por el sistema de coordenadas por defecto sólo se aproximan a los tamaños de los puntos como regla general y suelen estar predeterminadas a los píxeles de la pantalla para la mayoría de las pantallas.

javafx.scene.text.Text

La clase Texto define un nodo que muestra un texto. Los párrafos están separados por '\n' y el texto se envuelve en los límites del párrafo.

javafx.stage

Proporciona las clases contenedoras de nivel superior para el contenido de JavaFX.

javafx.stage.FileChooser

Proporciona soporte para los diálogos de archivos de la plataforma estándar. Estos diálogos tienen la apariencia de los componentes de la interfaz de usuario de la plataforma, que es independiente de JavaFX.

En algunas plataformas en las que el acceso a los archivos puede estar restringido o no formar parte del modelo de usuario (por ejemplo, en algunos dispositivos móviles o integrados), la apertura de un diálogo de archivo puede resultar siempre en un no-op (es decir, se devuelven archivos nulos).

Un FileChooser puede utilizarse para invocar diálogos de apertura de archivos para seleccionar un solo archivo (showOpenDialog), diálogos de apertura de archivos para seleccionar múltiples archivos (showOpenMultipleDialog) y diálogos para guardar archivos (showSaveDialog). La configuración del diálogo mostrado está controlada por los valores de las propiedades de FileChooser establecidas antes de llamar al método show*Dialog correspondiente. Esta configuración incluye el título del diálogo, el directorio inicial que se muestra en el diálogo y el/los filtro/s de extensión de los

archivos listados. Para las propiedades de configuración cuyos valores no se han establecido explícitamente, el diálogo mostrado utiliza sus valores por defecto de la plataforma. Una llamada a un método de mostrar diálogo se bloquea hasta que el usuario hace una elección o cancela el diálogo. El valor de retorno especifica el archivo(s) seleccionado(s) o es igual a null si el diálogo ha sido cancelado.

javafx.stage.ExtensionFilter

Define un filtro de extensiones, utilizado para filtrar qué archivos pueden ser elegidos en un FileDialog basado en las extensiones de los nombres de los archivos.

javafx.stage.Stage

La clase JavaFX Stage es el contenedor JavaFX de nivel superior. El Stage primario es construido por la plataforma. La aplicación puede construir objetos Stage adicionales.

Los objetos Stage deben construirse y modificarse en el hilo de la aplicación JavaFX.

Muchas de las propiedades de Stage son de sólo lectura porque pueden ser modificadas externamente por la plataforma subyacente y, por tanto, no deben ser vinculables.

Estilo

Un escenario tiene uno de los siguientes estilos:

- StageStyle.DECORATED - un escenario con un fondo blanco sólido y decoraciones de la plataforma.
- StageStyle.UNDECORATED - un escenario con un fondo blanco sólido y sin decoraciones.
- StageStyle.TRANSPARENT - un escenario con fondo transparente y sin decoraciones.
- StageStyle.UTILITY - un escenario con un fondo blanco sólido y decoraciones mínimas en la plataforma.

El estilo debe ser inicializado antes de que el escenario se haga visible.

En algunas plataformas las decoraciones pueden no estar disponibles. Por ejemplo, en algunos dispositivos móviles o integrados. En estos casos se aceptará una solicitud de ventana DECORADA o UTILIDAD, pero no se mostrarán decoraciones.

Propietario

Un escenario puede tener opcionalmente una ventana propietaria. Cuando una ventana es el propietario de un escenario, se dice que es el padre de ese escenario. Cuando se cierra una ventana padre, se cierran todas sus ventanas descendientes. El mismo comportamiento encadenado se aplica para una ventana padre que está iconificada. Una etapa siempre estará encima de su ventana padre. El propietario debe ser inicializado antes de que el escenario se haga visible.

Modalidad

Un escenario tiene una de las siguientes modalidades:

- Modality.NONE - un escenario que no bloquea ninguna otra ventana.
- Modalidad.WINDOW_MODAL - una etapa que bloquea los eventos de entrada de todas las ventanas desde su propietario (padre) hasta su raíz. Su raíz es la ventana antecesora más cercana sin propietario.
- Modality.APPLICATION_MODAL - una etapa que bloquea los eventos de entrada de ser entregados a todas las ventanas de la misma aplicación, a excepción de los de su jerarquía de hijos.

Cuando una ventana es bloqueada por una etapa modal, su orden Z relativo a sus ancestros es preservado, y no recibe eventos de entrada ni eventos de activación de ventanas, pero continúa animando y renderizando normalmente. Tenga en cuenta que mostrar un escenario modal no necesariamente bloquea a quien lo llama. El método show() regresa inmediatamente independientemente de la modalidad del escenario. Utilice el método showAndWait() si necesita bloquear a la persona que llama hasta que el escenario modal se oculte (se cierre). La modalidad debe ser inicializada antes de que el escenario se haga visible.

javax.swing

Proporciona un conjunto de componentes "ligeros" (todo en lenguaje Java) que, en la medida de lo posible, funcionan igual en todas las plataformas.

javax.swing.JOptionPane

JOptionPane facilita la aparición de un cuadro de diálogo estándar que solicita a los usuarios un valor o les informa de algo. Para obtener información sobre el uso de JOptionPane, consulte Cómo hacer cuadros de diálogo, una sección de El tutorial de Java.

Aunque la clase JOptionPane puede parecer compleja debido al gran número de métodos, casi todos los usos de esta clase son llamadas de una sola línea a uno de los métodos estáticos showXxxDialog que se muestran a continuación:

Nombre del Método	Descripción
showConfirmDialog	Hace una pregunta de confirmación, como sí/no/cancelar.
showInputDialog	Solicita alguna aportación.
showMessageDialog	Cuéntale al usuario algo que ha sucedido.
showOptionDialog	La gran unificación de los tres anteriores.

Cada uno de estos métodos también viene en un sabor showInternalXXX, que utiliza un marco interno para mantener el cuadro de diálogo (ver JInternalFrame). También se han definido múltiples métodos de conveniencia - versiones sobrecargadas de los métodos básicos que utilizan diferentes listas de parámetros.

Todos los diálogos son modales. Cada método showXxxDialog bloquea a la persona que lo llama hasta que la interacción del usuario haya terminado.

Variables y métodos

private String path

Variable usada en MediaPlayerController.java para almacenar el path del archivo de video abierto.

File file = fileChooser.showOpenDialog(null);

path = file.toURI().toString();

---

private List<String> extensions

Variable que almacena todas las extensiones de archivo válidas para nuestra aplicación en un array.

---

extensions = Arrays.asList(".mp4", "*.3gp", "*.mkv", "*.MP4", "*.MKV", "*.3GP", "*.flv", "*.wmv");*

---

media, mediaplayer, mediaview

Variables objeto usadas para montar nuestra “vista” del video, es decir, variables que usamos para cargar el video seleccionado y mostrarlo en la ventana de nuestra aplicación.

Media media = new Media(path);

mediaPlayer = new MediaPlayer(media);

mediaView.setMediaPlayer(mediaPlayer);

Con la variable media cargamos el video a partir del path seleccionado. La variable media se la pasamos como parámetro a mediaplayer, la cual será usada para crear la vista (mediaview) que finalmente veremos en la aplicación.

private Slider volumeSlider

Variable objeto, crea el slider de volumen. Nos permite después de haberla configurado, tener la funcionalidad de bajar y subir el volumen del video cargado.

private Slider progressBar

Variable objeto, crea el slider de progreso del video. Nos permite después de haberla configurado, tener la funcionalidad de movernos hacia adelante o hacia atrás por la duración del video.

private void OpenFileMethod()

Método asignado al botón de “Abrir Archivo” el cual nos permite elegir que archivo de video se quiere abrir y también carga el video en la ventana. En resumen (ya que el código es visible en su debido documento), este método abre la ventana de elegir archivo al clicar el botón diciendo que extensiones son las permitidas, crea el media, mediaplayer y mediaview al elegir el archivo y saber su path, determina el width y el height, inicializamos los slider y controlamos sus posibles eventos. Finalmente iniciamos el video.

private void playVideo()

Método asignado al botón de “Play/Pause” el cual nos permite pausar y reanudar el video al hacer clic en dicho botón. En resumen (ya que el código es visible en su debido documento), este método comprueba si el video está pausado o no y en función del estado hace un mediaplayer.play() o un mediaplayer.pause().

private void stopVideo()

Método asignado al botón de “Stop” el cual nos permite detener el video al hacer clic en dicho botón. En resumen (ya que el código es visible en su debido documento), este método lleva a cabo la función stop() del mediaplayer.

private void skip5()

Método asignado al botón de “Avanzar Video” el cual nos permite avanzar 5 segundos el video al hacer clic en dicho botón. En resumen (ya que el código es visible en su debido documento), este método lleva a cabo la función:

mediaPlayer.seek(mediaPlayer.getCurrentTime().add(javafx.util.Duration.seconds(5)))

private void back5 ()

Método asignado al botón de “Atrasar Video” el cual nos permite atrasar 5 segundos el video al hacer clic en dicho botón. En resumen (ya que el código es visible en su debido documento), este método lleva a cabo la función:

mediaPlayer.seek(mediaPlayer.getCurrentTime().add(javafx.util.Duration.seconds(-5)))

private void furtherSpeedUpVideo()

Método asignado al botón de “Acelerar” el cual nos permite acelerar el video al hacer clic en dicho botón. En resumen (ya que el código es visible en su debido documento), este método lleva a cabo la función: mediaPlayer.setRate(2).

private void furtherSlowDown()

Método asignado al botón de “Ralentizar” el cual nos permite ralentizar el video al hacer clic en dicho botón. En resumen (ya que el código es visible en su debido documento), este método lleva a cabo la función: mediaPlayer.setRate(0.5).

private void normalSpeedVideo()

Método asignado al botón de “Normalizar” el cual nos permite resetear la velocidad del video al hacer clic en dicho botón. En resumen (ya que el código es visible en su debido documento), este método lleva a cabo la función: mediaPlayer.setRate(1).

private void openAcercaDe()

Método asignado al menuItem “Acerca De” el cual nos abre la ventana acerca de al hacer click en dicho menuItem. En resumen (ya que el código es visible en su debido documento), este método hace visible la ventana AcercaDe.fxml

private void handleKeyPressed()

Método que contiene todos los atajos de teclado que tiene la aplicación. Todas las combinaciones de teclas son CTRL más la tecla seleccionada para cada acción. Todos los botones anteriormente mencionados tienen un atajo de teclado que llevan a cabo la misma funcionalidad que sus respectivos métodos (los atajos de teclado están explicados en el manual de usuario que acompaña a la aplicación).