Capítulo 1

Introducción a JavaFX

JavaFX es el sucesor de Swing como toolkit en la creación de interfaces gráficas para aplicaciones de escritorio escritas en lenguaje Java. Su última versión, JavaFX 8, es distribuida junto al Java Development Kit y Java Runtime Enviroment. Incluye APIs para dibujo de primitivas gráficas en 2D, construcción y renderizado de figuras 3D, y manejo de archivos multimedia por mencionar algunas. También posee un basto conjunto de componentes (botones, tablas, listas, etc) para la creación de interfaces gráficas.

Además, la última versión integra el uso de estándares web para la personalización de la apariencia de los componentes gráficos, separando el diseño y la programación de la lógica de negocios de las aplicaciones.

Actualmente, Oracle está desarrolla y soporta JavaFX, y desde 2011 se volvió una tecnología Open Source.

Evolución de GUI Toolkits

Con el pasar del tiempo la necesidad de interfaces usuario-sistema ha evolucionado. Antaño, bastaba con interfaces que permitieran la comunicación mediante comandos, esto es comprensible debido a que los sistemas estaban destinados a ser usados por un público delimitado y para propósitos muy específicos. Actualmente, las computadoras para uso personal son algo cotidiano por lo que una aplicación que pretenda ser bien recibida por un amplio público necesita una interfaz con controles gráficos que simplifiquen su uso. Para tal propósito, las diferentes compañías que han estado a cargo del desarrollo de Java han equipado al Java Development Kit con distintas GUI Toolkits desde la segunda versión del lenguaje.

En esta sección se presenta un recuento muy condensado de las GUI Tookits que han acompañado al lenguaje Java a lo largo de su historia previo a la llegada de JavaFX. También se mencionan otras toolkits para Java desarrolladas por terceros y se presenta el caso particular de Adobe Flex para Adobe Flash con el propósito de tener una base comparativa más completa para contrastar con JavaFX.

La figura 1.1 muestra una red de la evolución de las GUI toolkits de Java y su relación con otras tecnologías.

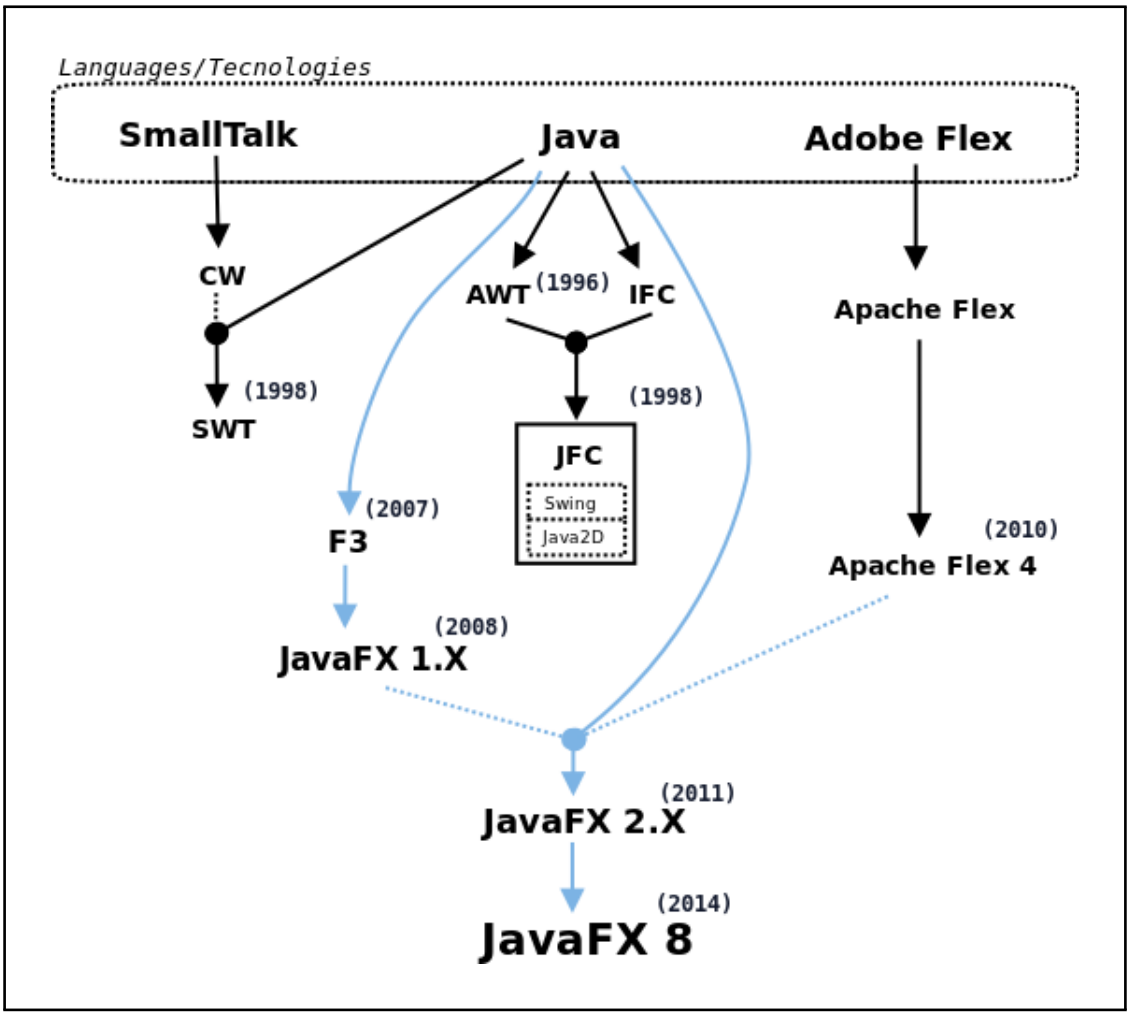

Figura 1.1: Evolución de las herramientas de creación de interfaces gráficas.

Abstract Window Toolkit (AWT) - 1996

- Primera toolkit para la creación de interfaces gráficas con Java.
- El renderizado de los componentes gráficos era realizado por el sistema operativo.
- Sus componentes gráficos se consideran componentes pesados pues al crearse un componente en código Java también se creaba un componente gráfico nativo por el sistema operativo.
- Apariencia (Look and Feel) variable dependiendo de la plataforma en que corriera la aplicación.
- Conjunto de componentes limitado a los componentes estándar admitidos por las diversas plataformas.

Internet Foundation Classes (IFC) - 1996

- Desarrollado por Netscape.
- Primera UI toolkit independiente de plataforma.
- El renderizado de los componentes era realizado por Java, no el SO.
- Soporte para Applets en el navegador Netscape.

Java Foundation Classes (JFC) - 1998

- Es resultado de la integración de IFC en Java.
- Da soporte para AWT.
- Contiene la Java2D API para el dibujo de primitivas gráficas.
- Se incluye Swing como nueva UI toolkit.
    - Amplio conjunto de componentes gráficos.
    - Componentes construidos con la arquitectura Modelo Vista Controlador.
    - Componentes ligeros pues Java se encarga completamente del renderizado.
    - Provee un API para la configuración del LAF de los componentes.

Java Foundation Classes (JFC) - 1998

- Es resultado de la integración de IFC en Java.
- Da soporte para AWT.
- Contiene la Java2D API para el dibujo de primitivas gráficas.
- Se incluye Swing como nueva UI toolkit.
    - Amplio conjunto de componentes gráficos.
    - Componentes construidos con la arquitectura Modelo Vista Controlador.

  - Componentes ligeros pues Java se encarga completamente del renderizado.
  - Provee un API para la configuración del LAF de los componentes.

Standard Widget Toolkit (SWT) - 1998

- Desarrollado por la Eclipse Foundation.
- Está basado en el IBM Common Widget Toolkit para el lenguaje SmallTalk.
- Posee un Look and Feel y desempeño nativo.
- Tal como lo hace AWT, SWT provee envolvedores al rededor de componentes nativos del SO.
- Los componentes que no son soportados por el SO son emulados con Java, similar al modo de Swing.

JavaFX 1.X - 2008

- Derivado del proyecto F3 se crea la primera versión de JavaFX.
- Desarrollado con el propósito de facilitar la creación de Rich Internet Applications.
- Incluye JavaFX Script, un lenguaje declarativo para definir interfaces gráficas.
- Facilita la creación de aplicaciones para las siguientes plataformas:
  - Escritorio
  - Móviles
  - Web
  - Televisores
  - Blu-ray

Apache Flex Spark- 2010

- Framework para la creación de clientes enriquecidos en lenguaje ActionScript (Flash).
- Posee la librería MXLM que hace uso de un archivo en formato XML para la definición de interfaces gráficas.
- Similar a MVC separa la vista en los archivos MXML y los controladores y modelos en código ActionScript.
- Soporta efectos y animaciones para los componentes de la interfaz.

JavaFX 2.X - 2011

- Finalizó el soporte a JavaFX Script, en su lugar se permite la creación de interfaces gráficas mediante APIs de Java y el uso de ficheros FXML.

- Se incluyó la capacidad de realizar animaciones, transformaciones y efectos en componentes de la interfaz gráfica.
- Fue añadida la interoperabilidad con Swing.
- Integración de un componente web que permite tener contenido HTML y ejecutar código JavaScript de forma embebida.

JavaFX 8 - 2014

- Soporte nativo para gráficas 3D.
- Soporte para uso de sensores.
- Versión incluida dentro de la edición estándar de Java.
- Fue añadida API para impresión.
- Nuevo conjunto de componentes gráficos añadidos.

Ventajas de JavaFX

Debido a su trayectoria histórica, JavaFX es el resultado de incorporar aquellos aspectos de provecho de sus predecesores y algunas virtudes de herramientas de GUI de otras tecnologías.

Partiendo del uso de una arquitectura MVC para la construcción de sus componentes gráficos podemos observar el panorama de ventajas que posee. Gracias a esta división entre la lógica y la presentación, se obtiene el benefició de mantenibilidad de las aplicaciones y se agiliza su desarrollo. La unión de la arquitectura MVC, el uso de archivos FXML en la definición de interfaces, la incorporación de hojas de estilo CSS para la personalización de la presentación de los componentes y las APIs para el empleo de archivos multimedia dota al desarrollador con la capacidad de crear interfaces que mejoren la experiencia del usuario.

Haciendo una comparativa con su predecesor Swing, JavaFX es superior en los siguientes aspectos.

Recursos: La cantidad de recursos requeridos para la ejecución de aplicaciones es menor debido a la incorporación de estrategias de optimización durante el renderizado.

Componentes: Tiene un conjunto de componentes gráficos y layouts más diverso y actualizado.

Usabilidad: Gracias a FXML, JavaFX simplifica cosas que con Swing y Java2D serían complejas de realizar.

3D: Da soporte a gráficas en 3D sin necesidad de alguna otra librería.

Por último, hay que recalcar que JavaFX es compatible completamente con JSE permitiendo al desarrollador usar todas aquellas APIs con las que está familiarizado.

Arquitectura de JavaFX

JavaFX posee una arquitectura compuesta de varios elementos, la figura 1.2 muestra su esquema arquitectónico. Algunos autores dividen su arquitectura simplemente en bloques y otros en capas, aquí se presenta la división en capas.

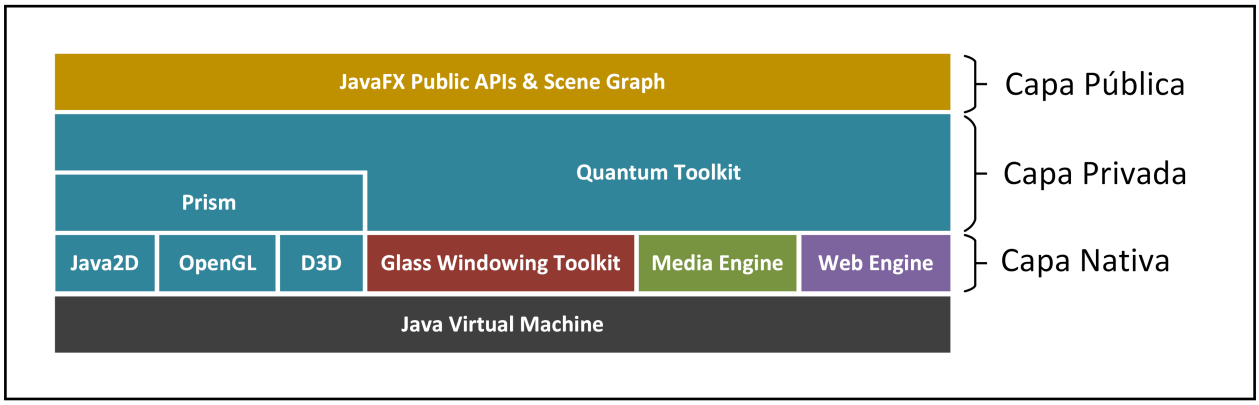

Figura 1.2: Arquitectura de JavaFX

Las Tres Capas

La capa nativa es la capa inferior de la arquitectura. Se compone (en su mayoría) de librerías no escritas en Java que dan acceso a la capa nativa del SO; entre estas librerías se encuentran D3D y OpenGL que son implementaciones de Prism, una tecnología para el renderizado mediante hardware o software de componentes gráficos.

También se incluyen motores para contenido web y multimedia, estos motores permiten embeber páginas HTML que hagan uso de CSS y JavaScript dentro de aplicaciones de JavaFX así como vídeos y música que enriquezca la experiencia del usuario. Otro componente de esta capa es el Glass Window Toolkit; este componente es el más bajo en el stack gráfico de JavaFX, entre otras cosas su principal función es proveer servicios operativos nativos como la administración de ventanas, temporizadores, superficies y la cola de eventos.

Debido a que los componentes nativos son específicos para cada SO algunas características de la capa nativa están condicionadas a la disponibilidad de estos componentes en la plataforma. Se puede comprobar la disponibilidad de estas características mediante código.

```
// Enumerado con todas las caracteristicas opcionales.
javafx.application.ConditionalFeature;

// Metodo que comprueba si una caracteristica es soportada.
Platform.isSupported(...);
```

En la capa privada se encuentra el sistema gráfico de JavaFX. Dos aceleradores forman este sistema: el primero es Prism, que como se mencionó antes, es encargado de los trabajos de renderizado vía hardware y software de las escenas de JavaFX; el segundo es el Quantum Toolkit cuya tarea es ligar Prism con el Glass Windowing Toolkit y hacerlos disponibles para la capa superior.

Tabla 1.1: Paquetes provistos en la API pública de JavaFX

Paquete	**Descripción**
javafx.animation	Contiene clases para el uso de animaciones basadas en transiciones
javafx.application	Provee las clases del ciclo de vida de la aplicación
javafx.beans	Contiene las clases que definen las API para hacer las propiedades vinculables a cambios
javafx.collections	Colecciones y utilidades esenciales para observar y reaccionar a cambios en el contenido de las colecciones
javafx.concurrent	Clases de ayuda para el manejo de procesos asíncronos
javafx.css	Provee APIs para hacer que las propiedades de los componentes gráficos sean personalizables vía CSS
javafx.event	Clases para la definición y el manejo de eventos en componentes de la interface
javafx.fxml	Contiene clases para la manipulación de archivos fxml
javafx.geometry	Conjunto de clases para la definición y operación de formas 2D
javafx.print	Clases para la impresión de archivos
javafx.scene	Conjunto principal de clases para la construcción del Scene Graph
javafx.stage	Contiene clases de contenedores de .ªlto nivelçomo ventanas o pop-ups
javafx.util	Clases de utilidad y ayuda

La capa pública es la más importante para el desarrollador, en ella se encuentran las APIs necesarias para el desarrollo de aplicaciones. En este mismo nivel se incluye el Scene Graph como método de construcción/representación de las interfaces gráficas de usuario. La tabla 1.1 lista todos los paquetes de la API pública y da una breve descripción de cada uno.

Scene Graph

El Scene Graph o grafo de escena es una representación jerárquica de la interface de usuario de la aplicación. Para cada ventana que tenga la aplicación existirá un grafo de escena. El grafo contiene nodos, estos son todos los elementos visuales de la interface: controles, primitivas gráficas, layouts, imágenes, etc. Cada nodo tiene una clase asociada a su estilo y un identificador único. Todos los nodos con excepción de la raíz poseen un padre y cero o más nodos hijos.

Usar este tipo de representación tiene dos ventajas, la primera es que se pueden aplicar efectos, transformaciones y escuchadores de eventos a nodos particulares o a un sub-árbol de nodos del grafo a la vez; la segunda es que se mejora el desempeño de la aplicación usando estrategias de optimización de renderizado de la escena.

Primer Ejemplo

El siguiente código servirá como acercamiento al Scene Graph, ciclo de vida de una aplicación y las APIs disponibles para la creación de interfaces.

Código 1.1: Código de la primera aplicación

```
1  import javafx.application.Application;
2  import javafx.scene.Scene;
3  import javafx.scene.control.Label;
4  import javafx.scene.layout.StackPane;
5  import javafx.stage.Stage;
6
7  public class App extends Application {
8
9     @Override
10    public voidvoid start(Stage primaryStage) throws Exception {
11             Label label = new Label("First␣application␣in␣JavaFX");
12
13             StackPane pane = new StackPane();
14             pane.getChildren().add(label);
15
16             Scene myScene = new Scene(pane);
17
18             primaryStage.setScene(myScene);
19             primaryStage.setWidth(400);
20             primaryStage.setHeight(300);
21             primaryStage.show();
22       }
23
24       public static void main(String[] args) {
25             launch(args);
26       }
27 }
```

La figura 1.3 muestra la ventana creada con el código anterior. Como se observa, ésta simplemente tiene un texto en ella, igual de sencillo será su grafo de escena pero para que se pueda comprender la estructura del grafo con claridad primero se explicará el código.

Las líneas 1-5 son las las sentencias de importación de las clases utilizadas de la API de JavaFX. Las clases Application, Stage y Scene son esenciales para elaborar una aplicación JavaFX.

En la línea 7 la clase principal extiende de *Application*. Extender de *Application* es requisito para que el método *main* tenga acceso al método *launch* y a los distintos métodos del ciclo de vida de las aplicaciones JavaFX.

En la tabla 1.2 se muestran los métodos del ciclo de vida de las aplicaciones JavaFX. La figura 1.4 ilustra el flujo de ejecución del ciclo de vida.

Desde la línea 10 a la 22 se sobreescribe el método abstracto *start(Stage)*, es en este método donde se construye el grafo de escena de nuestra aplicación. En la línea 11 se crea una instancia de la clase Label, en la línea 13 una de la clase administradora de diseño StackPane y en la línea 14 se añade el objeto label a los nodos hijo del objeto pane.

En la línea 16 se crea una instancia del grafo de escena y se añade el objeto del panel administrador de diseño, hasta este punto ya está construido el grafo de escena de la interface.

En la línea 18 se vincula el grafo de escena al objeto *primaryStage*. Este objeto fue suminis-

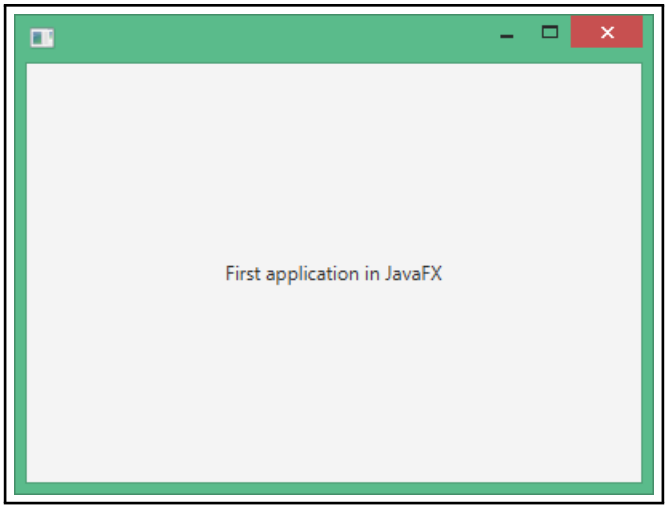

Figura 1.3: Ventana de la primera aplicación

Tabla 1.2: Métodos del ciclo de vida

Método	**Descripción**
launch(String[])	Método estático invocado desde el método *main*, inicia el ciclo de vida.
init()	Usado para inicializar los recursos que la aplicación pudiera necesitar.
start(Stage)	Añade el grafo de escena a la ventana de la aplicación.
stop()	Usado para cerrar conexiones a bases de datos, flujos a archivos y todo aquello que deba ser finalizado antes de terminar la ejecución de la aplcación.

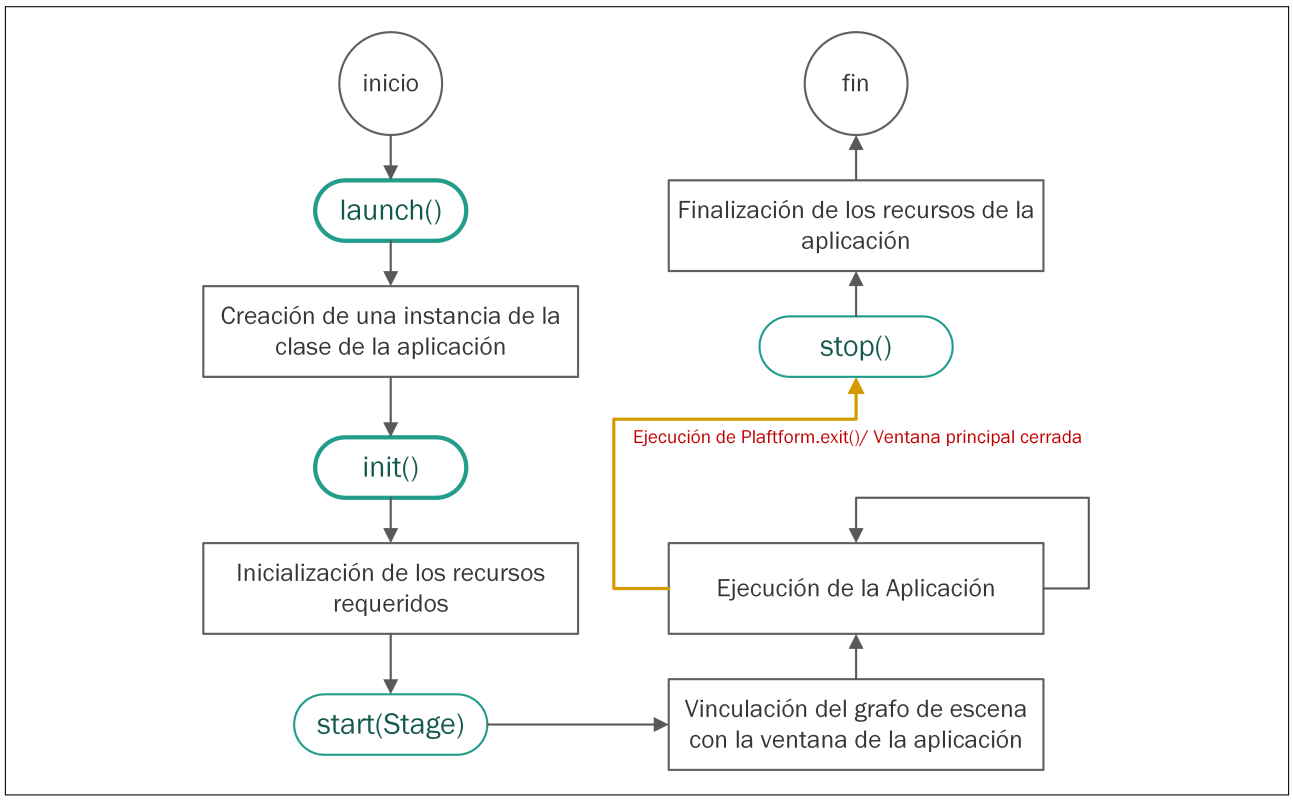

Figura 1.4: Flujo del ciclo de vida de una aplicación JavaFX

trado como un argumento por el ciclo de vida de la aplicación; puede considerarse a este objeto como la ventana principal de la aplicación.

Por último del método *start*, en las líneas 19 y 20 se especifica el tamaño de la ventana y en la 21 se hace visible.

Las líneas 24, 25 y 26 está la definición del método *main* de la aplicación, aquí es donde se inicia el ciclo de vida de la aplicación de JavaFX con el método *launch*.

La figura 1.5 muestra el grafo de escena del código 1.1, como se había mencionado este es un grafo muy simple debido a la sencillez de la aplicación, lo importante es comprender que el grafo de escena es una estructura jerárquica donde todos los elementos de la interface están contenidos, incluso los administradores de diseño.

De forma general se pueden definir los siguientes pasos para la creación de una aplicación con JavaFX.

1. Extender la clase *javafx.application.Application*.
2. Implementar el método *start(Stage)* y crear en él el grafo de la escena.
3. (Opcional) Implementar los métodos *init()* y *stop()* si la aplicación lo requiere.
4. Invocar en el método *main()* el método *launch()* de la subclase de *javafx.application.Application*.

En capítulos posteriores se explicará al lector con mayor detalle cuáles son los controles

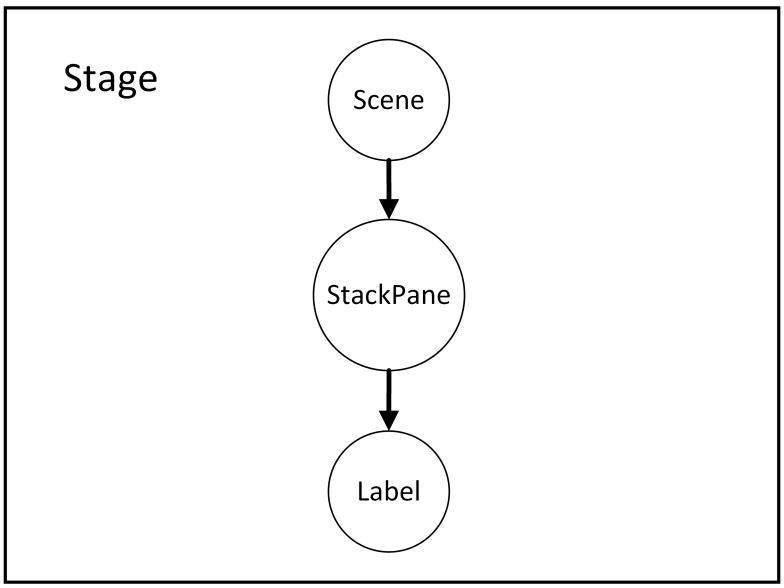

Figura 1.5: Grafo de escena del Código 1

disponibles, como construir interfaces mediante archivos FXML, graficación de figuras 2D, aplicaciones que operen mediante una red y aplicaciones con acceso a bases de datos.