

TECNOLÓGICO NACIONAL DE MÉXICO  
INSTITUTO TECNOLÓGICO DE VERACRUZ

REPORTE FINAL  
**DESARROLLO DE APLICACIONES CON  
JAVAFX UTILIZANDO LA  
ARQUITECTURA MVC**

Reporte final de actividades de servicio social

elaborado por  
**VICTOR DANIEL REBOLLOSO DEGANTE**  
Estudiante de Ingeniería en Sistemas Computacionales  
con número de control E12020839

supervisado por  
**MC. RAFAEL RIVERA LÓPEZ**  
Profesor - Investigador, Departamento de Sistemas y Computación  
Instituto Tecnológico de Veracruz

INGENIERÍA EN SISTEMAS COMPUTACIONALES

14 de agosto de 2017

# Índice general

<b>I</b>	<b>Marco Metodológico . . . . .</b>	<b>1</b>
1	Antecedentes . . . . .	1
2	Definición del Problema . . . . .	2
3	Objetivos . . . . .	2
3.1	Objetivo General . . . . .	2
3.2	Objetivos Específicos . . . . .	2
3.3	Justificación . . . . .	3
3.4	Alcance y Limitaciones . . . . .	3
<b>II</b>	<b>Marco Teórico . . . . .</b>	<b>4</b>
1	Java . . . . .	4
2	Arquitectura Modelo-Vista-Controlador . . . . .	4
3	Evolución de GUI Toolkits . . . . .	5
4	JavaFX . . . . .	9
4.1	Arquitectura . . . . .	9
5	eXtensible Markup Language . . . . .	11
5.1	Documentos Bien Formados . . . . .	12
5.2	Estructura Lógica . . . . .	12
6	FX Markup Language . . . . .	13
7	Cascading Style Sheets . . . . .	16
7.1	CSS & FXML . . . . .	16
<b>III</b>	<b>Desarrollo . . . . .</b>	<b>18</b>
1	Enfoque metodológico . . . . .	18
2	Identificación de la información pertinente . . . . .	18
3	Procedimiento . . . . .	19
<b>IV</b>	<b>Resultados . . . . .</b>	<b>20</b>
1	Material Didáctico en PDF . . . . .	20
2	Códigos de Ejemplo . . . . .	20
<b>V</b>	<b>Evidencias . . . . .</b>	<b>21</b>
<b>VI</b>	<b>Conclusiones . . . . .</b>	<b>23</b>
<b>VII</b>	<b>Recomendaciones . . . . .</b>	<b>24</b>
	<b>Anexos . . . . .</b>	<b>26</b>

<b>A</b>	<b>Introducción a JavaFX . . . . .</b>	<b>27</b>
<b>B</b>	<b>Elementos de Interfaces Gráficas . . . . .</b>	<b>39</b>
<b>C</b>	<b>Librerías de Interfaces Gráficas . . . . .</b>	<b>48</b>
<b>D</b>	<b>Computación Gráfica 1 . . . . .</b>	<b>73</b>
<b>E</b>	<b>Computación Gráfica 2 . . . . .</b>	<b>84</b>
<b>F</b>	<b>Computación Gráfica 3 . . . . .</b>	<b>90</b>
<b>G</b>	<b>Programación Orientada a Eventos . . . . .</b>	<b>99</b>

## Índice de figuras

II.1	Evolución de las herramientas de creación de interfaces gráficas. . . . .	6
II.2	Arquitectura de JavaFX . . . . .	10
II.3	Captura de pantalla a la interfaz del chat . . . . .	14
II.4	Captura de pantalla a la interfaz del chat con estilos modificados por CSS .	16
V.1	Captura de pantalla de evidencia de la elaboración del material con el serv- idor social, victor Rebolloso, en pantalla . . . . .	21
V.2	Captura de pantalla de evidencia de la aplicaciones elaboradas con el servi- dor social, victor Rebolloso, en pantalla . . . . .	22

## Índice de tablas

II.1	Paquetes provistos en la API pública de JavaFX . . . . .	11
IV.1	Archivos PDF del material didáctico . . . . .	20

# Índice de códigos

II.1	Ejemplos de elemento, elemento vacío y elementos con atributos . . . . .	13
II.2	Ejemplos de elemento, elemento vacío y elementos con atributos . . . . .	15
II.3	Estilo CSS para modificar la apariencia de los componentes de la GUI . . .	17

# Capítulo I

## Marco Metodológico

### 1. Antecedentes

Todo programa de computadora necesita un medio para que los usuarios puedan interactuar y hacer uso de él. Antaño, acorde a las necesidades del momento, las interfaces usuario-máquina de los programas eran simples líneas de comando en las que un usuario introducía una instrucción al programa en forma de texto y en consecuencia se ejecutaba la rutina programada. Las interacciones eran sencillas.

Al pasar el tiempo las necesidades sobre los sistemas computacionales fueron cambiando, las expectativas de formas más rápidas y visuales de utilizar los programas aumentaron y así llegaron los primeros sistemas operativos con interfaces gráficas. A causa de ello los programas tuvieron que adaptarse a los nuevos entornos en los que serían ejecutados. Se consiguieron ventanas con componentes visuales que organizaban la información que contenían y además ofrecían un comportamiento funcional como seleccionar una opción, recibir entradas de texto o números, mostrar avisos informativos o de advertencia, etc.

El tiempo continúo su curso y las tecnologías fueron avanzando con él. Mejoras estéticas y componentes visuales más complejos fueron generándose. Ahora los usuarios requieren facilidad para aprender a interactuar con los programas, que las interfaces sea intuitivas.

Java es un lenguaje con el que se pueden desarrollar aplicaciones de escritorio para diversos fines. Fue lanzado en 1995 y desde su inicio hasta hoy en día goza de popularidad y preferencia de desarrolladores y empresas para utilizarlo en sus proyectos.

Desde su primera versión posee una herramienta para la creación de interfaces de usuario y esta ha ido mejorando y cambiando entre versiones. Java 8 fue liberado en el año 2014 y actualmente es la versión estable más reciente del lenguaje. Entre las mejoras incluidas en esta versión se encuentra JavaFX, un framework para la construcción de Interfaces de Gráficas de Usuario (GUIs) para aplicaciones en Java.

Para los programadores que utilizan tecnología Java, JavaFX es un cambio relevante. Su importancia radica en que es un paso hacia la modernización de los métodos de construcción de GUIs luego de los casi 19 años de permanencia de las herramientas Swing y AWT, JavaFX fue creado con el propósito de sustituir a sus predecesoras.

El plan ISC-2010-224 de la carrera Ingeniería en Sistemas Computacionales en el Instituto Tecnológico de Veracruz incluye varias materias que preparan al estudiante en diversos temas y métodos para elaborar programas de computadora.

Tópicos Avanzados de Programación (TAP) es una materia que corresponde al cuatro semestre de la carrera de ISC, durante el curso se ven temas como Interfaces Gráficas de Usuarios, Computación Gráfica, Programación Orientada a Eventos y Concurrencia entre otros. Los cursos de TAP son independientes de cualquier lenguaje de programación, sin embargo, es común que se imparten utilizando el lenguaje Java para llevar la parte práctica de los temas, tal es el caso del MC. Rafael Rivera López quien ha impartido el curso utilizando Java durante varios años.

Hasta la fecha sólo se han utilizado versiones previas a Java 8 en los cursos y en consecuencia, poco se ha visto de las nuevas características del lenguaje su nueva herramienta JavaFX.

## 2. Definición del Problema

Dentro del Instituto Tecnológico de Veracruz se utiliza material didáctico que explica temas sobre la creación de interfaces gráficas de usuario para aplicaciones Java utilizando Swing, una herramienta funcional y útil pero que se pretende sea sustituida por JavaFX. Hasta la fecha no existe un material que explique los mismos temas utilizando la herramienta JavaFX.

## 3. Objetivos

### 3.1. Objetivo General

Explorar las capacidades de la tecnología JavaFX en la construcción de interfaces gráficas de usuario para aplicaciones Java y evaluar la viabilidad de su integración como herramienta de enseñanza en los cursos de Tópicos Avanzados de Programación impartido en actual plan de estudios de la carrera de Ingeniería en Sistemas Computacionales en el Instituto Tenológico de Veracruz.

### 3.2. Objetivos Específicos

Como objetivos particulares de este trabajo se establecieron los siguientes:

1. Recopilar literatura referente a la tecnología JavaFX que sirva como referencia para desarrollar el resto de objetivos del proyecto.
2. Elaborar documentos que expliquen los temas:
  - Elementos de una interface gráfica,
  - Librerías de Interfaz Gráfica,
  - Computación Gráfica,
  - Programación Orientada a Eventos y
  - Arquitectura MVC.

utilizando JavaFX y que puedan ser empleados en la enseñanza de los temas mencionados en la materia Tópicos Avanzados de Programación.

3. Crear un conjunto de códigos de programación que ejemplifiquen de forma práctica los temas mencionados previamente.

### 3.3. Justificación

Se obtendrá material didáctico sobre temas importantes en la formación de los estudiantes de la carrera de Ingeniería en Sistemas Computacionales (ISC) explicados con la nueva herramienta de construcción de GUIs para aplicaciones de uno de los lenguajes de programación más utilizados en el ámbito académico y solicitados en el ejercicio de una profesión.

Se ofrece una alternativa nueva para la enseñanza de TAP, temas que hasta la fecha se han explicado con otras tecnologías como Swing de Java, la actual opción predilecta. Un motivo importante de la creación de este material es que Swing, a pesar de su permanencia y disponibilidad para los programadores, es una herramienta que ya no recibirá más actualizaciones por parte de Oracle, siendo JavaFX la nueva apuesta para la creación de interfaces gráficas de usuario.

Los beneficiados inmediatos son los estudiantes de ingeniería en sistemas computacionales pues tienen una opción más fresca para aprender la construcción de interfaces gráficas de usuario con el lenguaje Java; beneficiados secundarios son todos aquellos interesados en conocer las cualidades y capacidades básicas del framework JavaFX.

El producto obtenido no es de provecho sólo por el hecho de ser un material que habla de una herramienta nueva (JavaFX), sino que esta herramienta fue creada por Oracle con el objetivo de **facilitar las actividades de desarrollo de las interfaces gráficas** incorporando esquemas manejados por otras tecnologías, y **mejorar su desempeño** gracias a estrategias de optimización de renderizado de los componentes gráficos.

### 3.4. Alcance y Limitaciones

El desarrollo del material se limitará sólo a la primera unidad temática del actual curso de Tópicos Avanzados de Programación impartido por el profesor MC. Rafael Rivera López.

Se tomará el material actual del MC. Rafael Rivera como base para su actualización a JavaFX. Se sustituirán directamente los temas que no necesiten explicación a detalle sobre JavaFX, se añadirán explicaciones complementarias en los temas que lo requieran y excluirán los temas que ya no apliquen en la nueva tecnología.

# **Capítulo II**

## **Marco Teórico**

### **1. Java**

Java fue lanzado en 1995 por la compañía Sun Microsystems como un lenguaje de programación orientado a objetos para múltiples plataformas. Sun estuvo encargado de su desarrollo y soporte hasta el 2010, año en que fue adquirido por Oracle. Su última versión, Java 8, fue liberada en marzo del 2014 y en ella se añadieron mejoras al lenguaje como algunas características del paradigma de programación funcional (uso de flujos de colecciones, interfaces funcionales y su versión simplificada las funciones lambda por ejemplo), además de que se incluyó JavaFX para la creación de GUIs.

El eslogan de Java fue durante mucho tiempo WORA, un acrónimo de la frase en inglés Write Once Run Anywhere (escribe una vez y corre/ejecuta donde sea), que hace referencia a su calidad de lenguaje multiplataforma. Esta característica es posible debido a la Java Virtual Machine, el motor que permite ejecutar código escrito en Java en cualquier computadora sin importar su arquitectura física.

Actualmente con Java es posible crear sistemas para escritorio usando Swing y JavaFX, para plataformas web con el estándar JEE y para plataformas móviles con el Software Development Kit de Android.

### **2. Arquitectura Modelo-Vista-Controlador**

Un patrón arquitectónico es una descripción abstracta de una arquitectura de software que ha sido utilizada exitosamente y que es producto de buenas prácticas de desarrollo de software. En esencia, el patrón de diseño Modelo Vista Controlador (MVC) consiste en dividir conceptualmente la arquitectura de un sistema y, en la medida de lo posible, la implementación de la misma en componentes que pertenezcan a una de tres categorías, Modelo, Vista y Controlador.

Dentro de la categoría de Modelo se ubican aquellos componentes encargados de: encapsular los datos utilizados en el sistema, representar el comportamiento de las entidades involucradas en los procesos de negocio y realizar tareas de persistencia de datos.

La categoría de Vista contiene los componentes con los que el usuario interactúa directamente, comúnmente también se les refiere como el front-end del sistema. Es importante notar la independencia entre la vista y el modelo. Modificaciones en los componentes que pertenecen a la vista no necesariamente implican modificaciones en el modelo y viceversa.

Por último, los componentes pertenecientes a la categoría Controlador se encargan de atender las peticiones del usuario solicitadas a través de los componentes de la Vista haciendo uso de los componentes del Modelo. Los controladores son el puente entre el Modelo y la Vistas del sistema.

Algunos de los beneficios de desarrollar un sistema bajo el patrón MVC son:

- Simplifica el desarrollo brindando un desglose su arquitectura.
- Separa las distintas responsabilidades (lógica de negocios, interfaz de usuario) de sus componentes.
- Facilita su mantenibilidad.

### 3. Evolución de GUI Toolkits

Con el pasar del tiempo la necesidad de interfaces usuario-sistema ha evolucionado. Actualmente, las computadoras para uso personal son algo cotidiano y el alcance de las aplicaciones es muy extenso, 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.

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 II.1 muestra una red de la evolución de las GUI toolkits de Java y su relación con otras tecnologías.

#### 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.

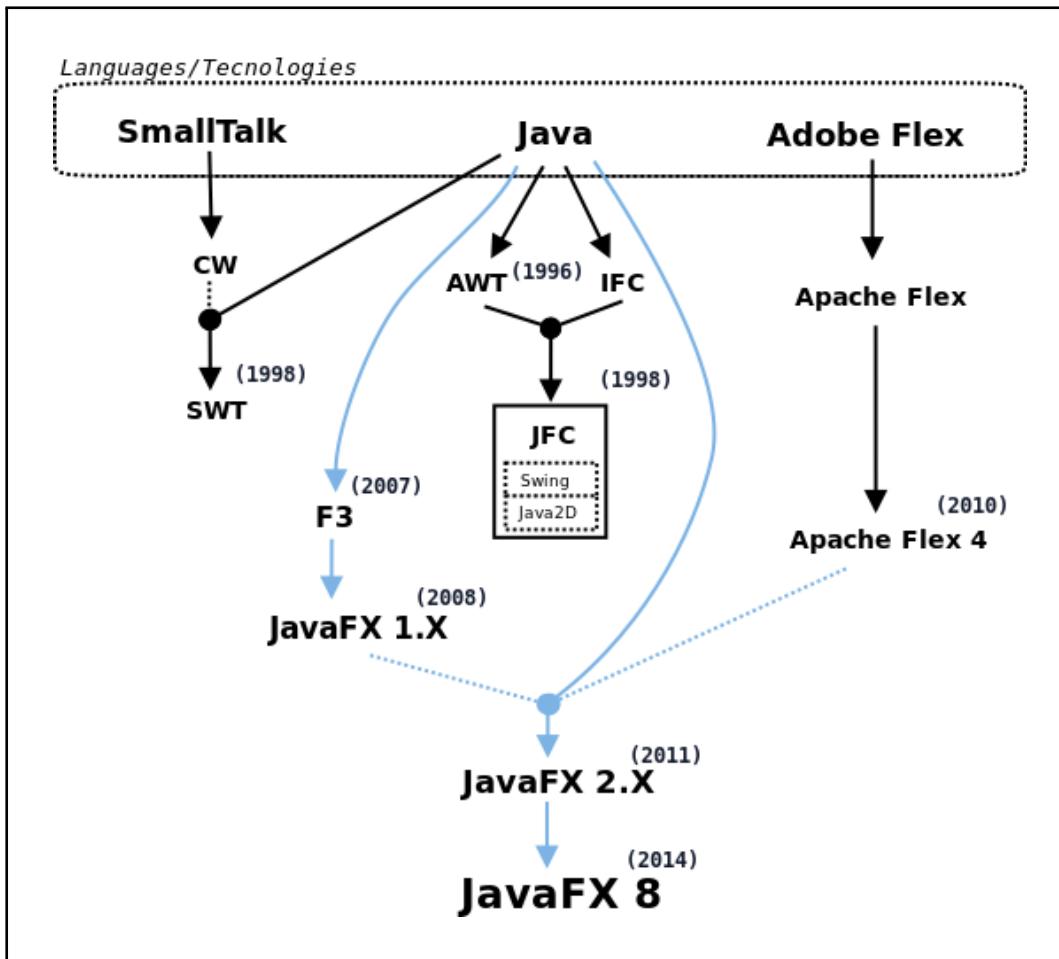


Figura II.1: Evolución de las herramientas de creación de interfaces gráficas.

- 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 MXML 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.

## 4. 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 Environment. 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 desarrolla y soporta JavaFX, y desde 2011 se volvió una tecnología Open Source.

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. JavaFX facilita la creación de aplicaciones bajo la arquitectura MVC y debido a la división entre la lógica y la presentación, se obtiene el beneficio 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.

### 4.1. Arquitectura

JavaFX posee una arquitectura compuesta de varios elementos, la figura II.2 muestra su esquema arquitectónico. Algunos autores dividen su arquitectura simplemente en bloques y otros en capas, aquí se presenta la división en capas.

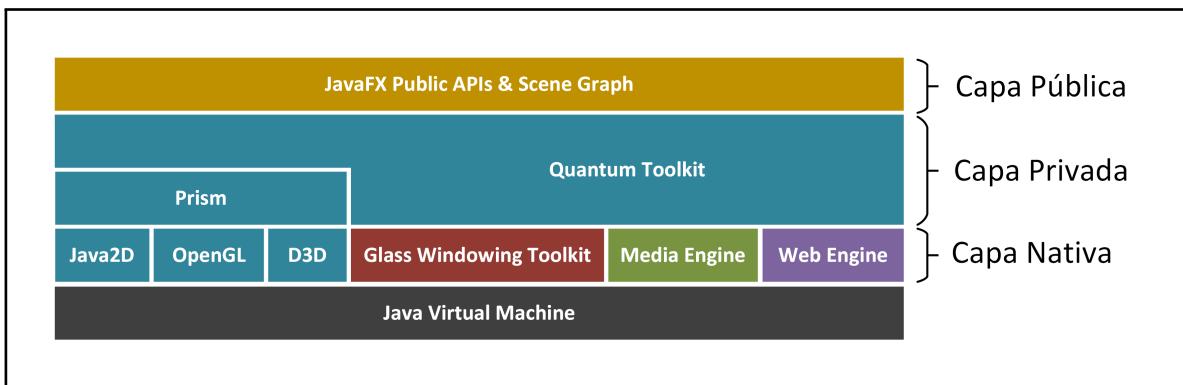


Figura II.2: Arquitectura de JavaFX

### Capa Nativa

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.

### Capa Privada

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.

### Capa Pública

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 II.1 lista todos los paquetes de la API pública y da una breve descripción de cada uno.

Tabla II.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 . <sup>a</sup> lto nivel como ventanas o pop-ups
javafx.util	Clases de utilidad y ayuda

## 5. eXtensible Markup Language

El lenguaje extensible de marcado, abreviado XML, es un lenguaje basado en etiquetas similar a HTML cuyo propósito es describir objetos de datos llamados documentos XML para almacenar, transportar, y en general, compartir datos a través entre distintas aplicaciones y sistemas.

El lenguaje fue desarrollado en 1996 por un grupo patrocinado por el World Wide Web Consortium, una comunidad internacional que desarrolla estándares abiertos para asegurar y promover el crecimiento a largo plazo de la Web.

Estas son la metas con que fue diseñado XML:

1. Debe ser fácilmente utilizable sobre internet.
2. Debe soportar una amplia variedad de aplicaciones.
3. Debe ser compatible con SGML (Standard Generalized Markup Language) un estándar para definir lenguajes de marcado.
4. Debe ser sencillo escribir programas que procesen documentos XML.
5. El número de características opcionales en XML debe mantenerse el su mínimo absoluto, idealmente cero.
6. Los documentos XML deben ser legibles para los humanos y razonablemente claros.
7. El diseño de XML debe ser preparado rápidamente.
8. El diseño de XML debe ser formal y conciso.
9. Debe ser fácil crear documentos XML.
10. La brevedad en el etiquetado XML es de mínima importancia.

## 5.1. Documentos Bien Formados

Se considera que un documento XML está bien formado si cumple con los siguientes puntos:

- Sólo contiene un elemento raíz que es el inicio de la estructura del documento.
- Si una etiqueta de apertura se encuentra dentro de un elemento A su correspondiente etiqueta de cierre debe estar dentro del mismo elemento A.
- Cumple con las reglas sintácticas para definir un documento XML.

En consecuencia cualquier documento bien formado podrá ser validado y manipulado por cualquier procesador XML.

## 5.2. Estructura Lógica

Los documentos XML se componen esencialmente de tres tipos de componentes: elementos, elementos vacíos y atributos.

**Elemento:** Un elemento tiene asignado un nombre para la etiqueta que lo representa. Los límites del elemento dentro de un documento están determinados por una etiqueta de apertura y otra etiqueta de cierre identificada con el nombre del elemento. Dentro de sí un elemento puede contener a cualquier otro elemento. Los elementos pueden tener atributos que lo describen, estos se ubican dentro de su etiqueta de apertura.

**Elemento vacío:** Este caso especial de elemento no contiene a ningún otro dentro de su cuerpo y por lo tanto no es necesario el uso de etiquetas de apertura y cierre, en su lugar se ocupa una etiqueta vacía. Los elementos vacíos pueden contener atributos.

**Atributos:** Los atributos dan información extra sobre el elemento en que se colocan. Se escriben siguiendo un formato clave-valor.

El código II.1 presenta ejemplos de los tres tipos de componentes descritos. Las líneas que se encuentran entre "

Código II.1: Ejemplos de elemento, elemento vacío y elementos con atributos

```

<!-- *** Elemento con contenido **** -->

<!-- Etiqueta de apertura -->
<carta>
  <!-- Elemento interno -->
  <destinatario>
    Pancha
  </destinatario>

  <remitente>
    Pedro
  </remitente>

  <mensaje>
    Estimada Pancha,

    ¿Cuánto tiempo ha pasado desde nuestra última carta? Últimamente he...
  </mensaje>
  <!-- Etiqueta de cierre -->
</carta>

<!-- *** Elemento vacío **** -->

<imagen />

<!-- *** Elemento con contenido y atributos **** -->

<carta fecha="22/12/2017" > <!-- Atributos -->
  ...
  <mensaje>
    ...
  </mensaje>
</carta>

<!-- *** Elemento vacío y con atributos **** -->

<imagen descripcion="La imagen." fuente="http://someaddress.com/imagen.png" />

```

XML puede ser extendido para definir un subconjunto del lenguaje con restricciones sintácticas más específicas. Comúnmente la especialización de este XML en un dialecto es para delimitar el layout de los documentos XML que manipulará una aplicación/sistema para un propósito muy particular, tal es el caso del lenguaje FXML utilizado por JavaFX.

## 6. FX Markup Language

FXML es un dialecto derivado de XML creado por Oracle para ser utilizado dentro de la tecnología de JavaFX. FXML es un lenguaje declarativo tal como XML pero su propósito no es almacenar información o ser usado para transmitir datos, su función es describir el diseño y disposición (layout) de los componentes de las interfaces gráficas de usuario. El dialecto es similar al esquema manejado en HTML con la diferencia que las etiquetas predefinidas en FXML corresponden, en su mayoría, a clases de Java incluida que representan controles y/o contenedores de los componentes gráficos.

El código II.2 es un ejemplo de una ventana de chat (véase figura II.3) elaborada

descrita en un documento FXML.

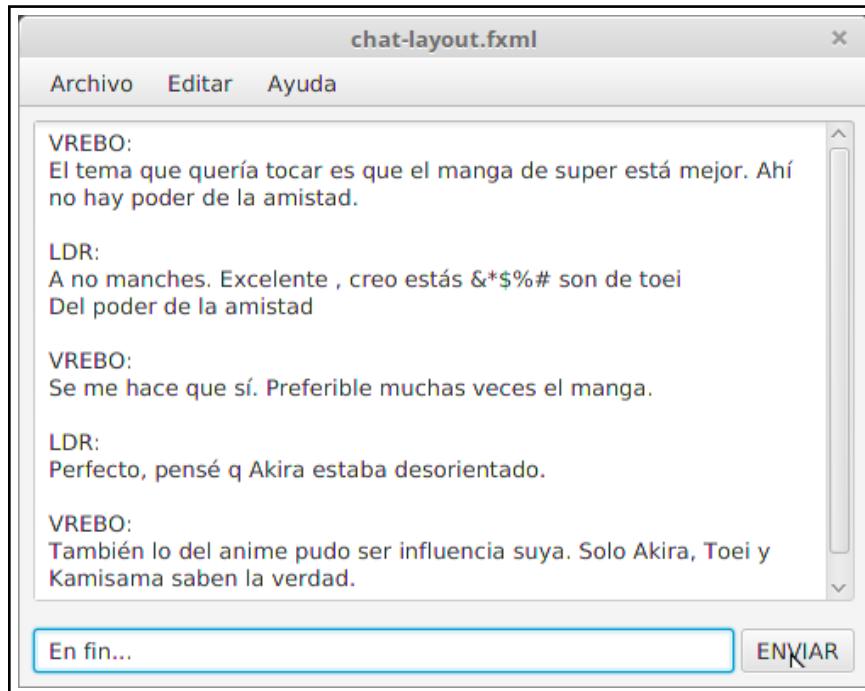


Figura II.3: Captura de pantalla a la interfaz del chat

Código II.2: Ejemplos de elemento, elemento vacío y elementos con atributos

```
<?xml version="1.0" encoding="UTF-8"?>
<?import javafx.geometry.Insets?>
<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Menu?>
<?import javafx.scene.controlMenuBar?>
<?import javafx.scene.control.MenuItem?>
<?import javafx.scene.control.SeparatorMenuItem?>
<?import javafx.scene.control.TextArea?>
<?import javafx.scene.control.TextField?>
<?import javafx.scene.layout.HBox?>
<?import javafx.scene.layout.StackPane?>
<?import javafx.scene.layout.VBox?>

<VBox maxHeight="-Infinity" maxWidth="-Infinity"
       minHeight="-Infinity" minWidth="-Infinity"
       prefHeight="400.0" prefWidth="600.0"
       xmlns="http://javafx.com/javafx/8.0.111" xmlns:fx="http://javafx.com/fxml/1">
    <children>
        <MenuBar>
            <menus>
                <Menu text="_Archivo">
                    <items>
                        <MenuItem mnemonicParsing="false" text="Abrir" />
                        <SeparatorMenuItem mnemonicParsing="false" />
                        <MenuItem mnemonicParsing="false" text="Salir" />
                    </items>
                </Menu>
                <Menu text="_Editar">
                    <items>
                        <MenuItem mnemonicParsing="false" text="Delete" />
                    </items>
                </Menu>
                <Menu text="A_yuda">
                    <items>
                        <MenuItem mnemonicParsing="false" text="About" />
                    </items>
                </Menu>
            </menus>
        </MenuBar>
        <StackPane VBox.vgrow="ALWAYS">
            <padding>
                <Insets bottom="8.0" left="8.0" right="8.0" top="8.0" />
            </padding>
            <children>
                <TextArea prefHeight="200.0" prefWidth="200.0" />
            </children>
        </StackPane>
        <HBox spacing="4.0" VBox.vgrow="NEVER">
            <children>
                <TextField promptText="Escribe tu mensaje..." HBox.hgrow="ALWAYS" />
                <Button mnemonicParsing="false" text="ENVIAR" />
            </children>
            <padding>
                <Insets bottom="8.0" left="8.0" right="8.0" top="8.0" />
            </padding>
        </HBox>
    </children>
</VBox>
```

## 7. Cascading Style Sheets

Las hojas de estilo en cascada (de su traducción del Inglés Cascading Style Sheets) son un mecanismo que permite añadir estilo a los documentos web. Su uso delega la responsabilidad de definir la presentación de los documentos a los archivos CSS y se evita la práctica poco recomendable de añadir estilos mediante los elementos del documento.

CSS es un lenguaje basado reglas, las cuales describen al navegador como presentar los elementos de HTML. A diferencia de otros lenguajes, CSS no tiene versiones, en su lugar tiene niveles siendo el último el nivel 3.

### 7.1. CSS & FXML

La tecnología CSS con XML y de la misma forma que se pueden dar estilos a los elementos en documentos HTML se puede hacer con FXML. JavaFX también define un conjunto de atributos para definir reglas de presentación visual que influirán en la apariencia de los componentes visuales de la interfaz gráfica.

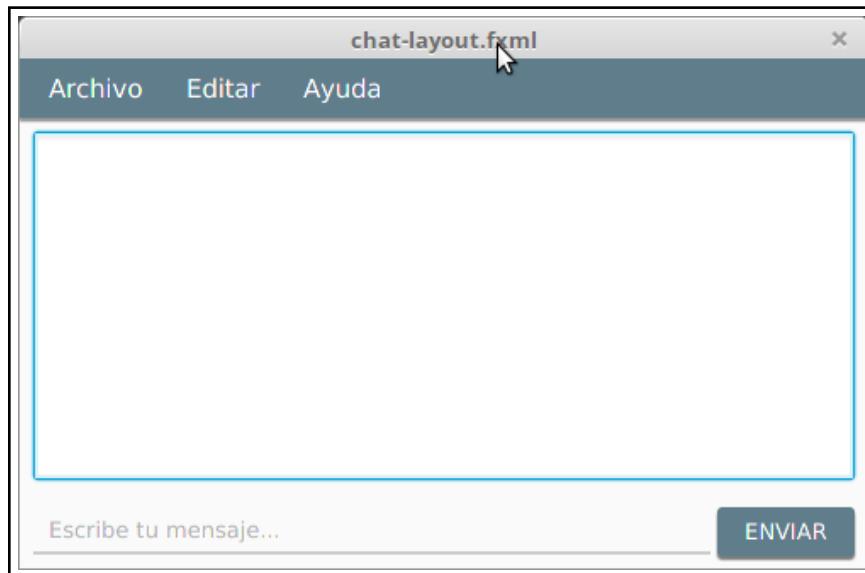


Figura II.4: Captura de pantalla a la interfaz del chat con estilos modificados por CSS

Código II.3: Estilo CSS para modificar la apariencia de los componentes de la GUI

```
.button {  
    -fx-effect: dropshadow(one-pass-box, rgba(0,0,0,0.5), 2, 0.3, 0, 2);  
    -fx-border-radius: 0px 0px 0px;  
    -fx-background-color: rgb(96,125,139);  
    -fx-padding: 6px 16px 6px 16px;  
    -fx-text-fill: white;  
}  
  
.menu-bar {  
    -fx-effect: dropshadow(one-pass-box, rgba(0,0,0,0.4), 2, 0.3, 0, 2);  
    -fx-background-color: rgb(96,125,139);  
    -fx-min-height: 20px;  
}  
  
.menu {  
    -fx-padding: 8px 8px 8px 8px;  
}  
  
.menu .label {  
    -fx-text-fill: white;  
    -fx-font-size: 11pt ;  
}  
  
.menu-item .label {  
    -fx-text-fill: black;  
    -fx-font-size: 9pt ;  
}  
  
.main-container {  
    -fx-background-color: #FAFAFA;  
}  
  
.text-field {  
    -fx-background-color: null;  
    -fx-border-color: rgba(0,0,0,0) rgba(0,0,0,0) #cccccc rgba(0,0,0,0);  
    -fx-border-width: 0 0 2px 0;  
}
```

# Capítulo III

## Desarrollo

### 1. Enfoque metodológico

Se considera el presente trabajo como un proyecto de investigación documental con enfoque pragmático orientado a la generación de material didáctico para la explicación los temas planteados en los objetivos guardando estrecha relación con la tecnología JavaFX.

La literatura utilizada para fundamentar el desarrollo de este trabajo proviene de fuentes editoriales oficiales como Apress y la propia compañía Oracle creadora de JavaFX. Otra fuente importante de información utilizada como complemento a la literatura editorial fueron los sitios web oficiales de la World Wide Web Consortium y Oracle. Por último, sitios de web de consulta para resolución de dudas y problemas de programación como Stackoverflow, CodeRanch, Quora, TexExchange, entre otros, fueron de gran ayuda en la elaboración del material didáctico.

### 2. Identificación de la información pertinente

Partiendo de los temas planteados en los objetivos del proyecto se seleccionó de la literatura recopilada aquella información relacionada a los siguientes temas:

- Historia de las GUIs Toolkits para la creación de interfaces gráficas de usuario en Java.
- Catálogo de componentes gráficos de las GUIs Toolkits de Java.
- Características de JavaFX.
- Construcción programática de GUIs (código Java).
- Construcción declarativa de GUIs (documentos FXML).
- Gráficas 2D con JavaFX.
- Esquema de JavaFX para el manejo de eventos en GUIs.
- Esquema MVC y JavaFX.

### 3. Procedimiento

Durante el ejercicio de las actividades de este proyecto de servicio social se siguió el procedimiento descrito a continuación:

1. Estudio del material actual de tópicos avanzados de programación del MC. Rafael Rivera López.
2. Estudio de la literatura recopilada de JavaFX.
3. Identificar de los temas y ejemplos clasificandolos con el siguiente criterio:

**Actualización Directa:** El tema o ejemplo se puede explicar con JavaFX directamente sin necesidad de explicación extra.

**Explicación Complementaria:** La actualización requiere añadir una explicación complementaria para asegurar la claridad del tema.

**Eliminación por Incompatibilidad:** El tema no se adapta al esquema utilizado por JavaFX o las actualizaciones son a tal grado que cambia significativamente el método de explicación original.

4. Compilación de la información y redacción de los temas actualizados.
5. Elaboración de diagramas e imágenes requeridas para ejemplificar los temas actualizados.
6. Elaboración de los códigos de ejemplo necesarios siguiendo el patrón MVC.
7. Integración de la información, diagramas y códigos de ejemplo en documentos PDF siguiendo la organización de los documentos de material didáctico originales.

El procedimiento anterior se realizó para cada uno de los archivos en formato PDF del material original correspondientes a la primera unidad, Elementos de Interfaces Gráficas, del curso de Tópicos Avanzados de Programación.

# Capítulo IV

## Resultados

Como resultado de este proyecto se obtuvo material didáctico en archivos con formato PDF y paquetes de códigos de ejemplos. El material se puede obtener en éste<sup>1</sup> vínculo.

### 1. Material Didáctico en PDF

En la tabla IV.1 se enlistan los archivos actualizados con los temas sobre JavaFX. La tabla presenta el nombre del archivo, una breve descripción y el anexo donde se puede encontrar.

Tabla IV.1: Archivos PDF del material didáctico

Archivo	Tema	Anexo
0 Introducción	Introducción a la tecnología JavaFX.	Anexo A (Pág. 27)
1.1 Elementos de Interfaces Graficas	Explicación de los Principales Componentes de una GUI.	Anexo B (Pág. 39)
1.2 Librerias de Interfaz Grafica	Presentación de las tres herramientas de Java para la creación GUIs.	Anexo C (Pág. 48)
1.3 Computacion Grafica	Introducción al método de gestión de gráficos y sus primitivas.	Anexo D (Pág. 73)
1.3 Computacion Grafica -B	Ejemplos de uso de Canvas y primitivas Gráficas.	Anexo E (Pág. 84)
1.3 Computacion Grafica -C	Ejemplo de uso de flujos de archivos para lectura y graficación de puntos.	Anexo F (Pág. 90)
1.4 Programación Orientada a Eventos	Explicación del esquema de gestión de eventos de GUIs.	Anexo G (Pág. 99)

### 2. Códigos de Ejemplo

Todos los códigos de ejemplo de los temas explicados se incluyeron en tres proyectos de lenguaje Java creados con el IDE Netbeans.

---

<sup>1</sup><https://github.com/vrebo/TAP-JavaFX>

# Capítulo V

## Evidencias

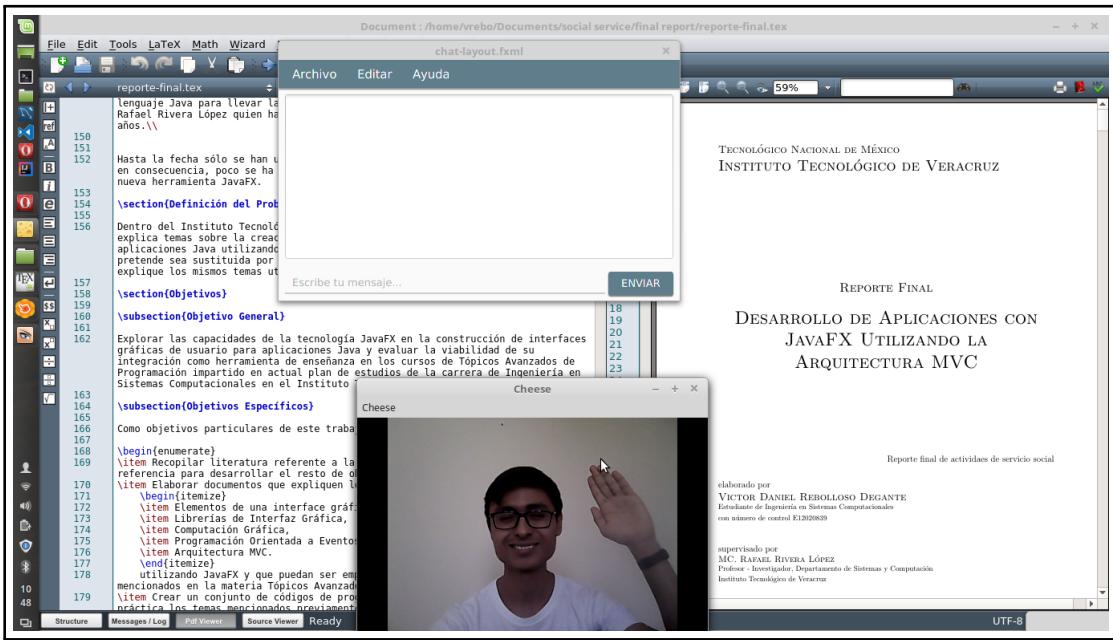


Figura V.1: Captura de pantalla de evidencia de la elaboración del material con el servidor social, victor Rebolloso, en pantalla.

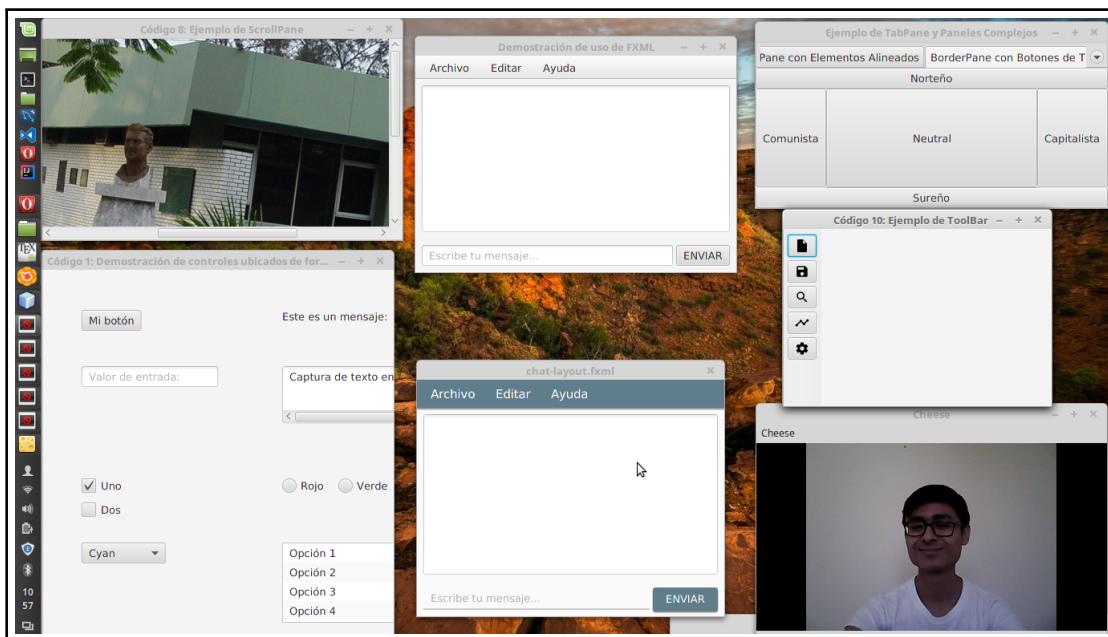


Figura V.2: Captura de pantalla de evidencia de la aplicaciones elaboradas con el servidor social, victor Rebolloso, en pantalla.

# Capítulo VI

## Conclusiones

En relación a los objetivos planteados se concluye lo siguiente:

- El objetivo general propuesto para este trabajo se logró. Se obtuvo material didáctico que ofrece un acercamiento inicial al desarrollo de interfaces gráficas de usuario utilizando JavaFX.
- Se exploraron las nuevas funciones de JavaFX como la definición de GUIs con documentos FXML y la modificación de su apariencia mediante hojas de estilos CSS.
- Sólo un tema de los propuestos a cubrir quedó fuera, Arquitectura MVC, pues no se explicó a detalle en un documento independiente. Sin embargo, en algunos párrafos de los documentos Introducción y Elementos de Interfaces Gráficas se hace mención sobre la características de JavaFX y cómo se adecúan en la arquitectura MVC. Además todos los códigos de programas de ejemplos se elaboraron siguiendo la división arquitectónica MVC.
- Se desarrollaron un conjunto de programas que ejemplifican los temas explicados en el material.

Y basado en la experiencia de este proyecto, a manera personal y expresado a modo de opinión, puedo concluir lo siguiente:

- La incorporación de tecnologías web con JavaFX es un paso en la dirección correcta para mejorar el paradigma actual de creación de interfaces de usuario. Estas tecnologías favorecen al programador pues reducen el tiempo y la complejidad de desarrollo de aplicaciones haciendo sencillo emplear una arquitectura MVC.
- A pesar de que JavaFX es la tecnología más reciente para GUIs ofrecida por Oracle han pasado 3 años desde su lanzamiento y su popularidad entre los desarrolladores en comunidades de internet es mucho menor de lo que se puede esperar. Lo siguiente es una especulación pero su baja popularidad puede deberse a la reciente proliferación de frameworks como Node.js y electron con los que se pueden hacer aplicaciones de escritorio con tecnología Javascript, CSS y HTML con relativa facilidad.
- Considerando únicamente el espectro de herramientas ofrecidas para elaborar interfaces de usuario para Java, JavaFX es la mejor opción.

# Capítulo VII

## Recomendaciones

Se presentan las siguientes recomendaciones para trabajos derivados de este proyecto:

- Incluir más ejemplos de creación de GUIs mediante documentos FXML.
- Incluir más ejemplos de modificación de la apariencia de GUIs mediante archivos CSS.
- Añadir a los temas de la primera unidad el uso de "Binding Properties", un mecanismo que permite observar cambios en algún atributo de un objeto y realizar alguna acción en respuesta a esos cambios, además puede ser usado para simplificar la transmisión de datos introducidos por una vista hacia su clase modelo.
- Continuar con la actualización de las unidades 2, Programación Concurrente y 3, Programación Móvil. Para la última unidad se sugiere utilizar JavaFXPorts que es un proyecto Open Source que da soporte para Java y JavaFX en plataformas móviles (Android, IOS) y en hardware embebido (Raspberry Pi).

# Bibliografía

- [1] Hendrik Ebbers, *Mastering JavaFX 8 Controls*, Mc Graw Hill, 1ra Edición, 2014.
- [2] Carl P. Dea, Mark Heckler, et. al., *JavaFX 8: Introduction by Example*, Apress, S.E, S.F.
- [3] Johan Vos, Weiqi Gao, et. al., *Pro JavaFX 8*, Apress, S.E, S.F.
- [4] Jasper Potts, Nancy Hildebrandt, et. al., *JavaFX: Getting Started with JavaFX Release 8*, Oracle, 1ra Edición, 2014.
- [5] Lawrence PremKumar & Praveen Mohan, *Beginning JavaFX*, Apress, S.E, 2010.
- [6] Doug Lowe, *JavaFX for Dummies*, John Wiley & Sons, Inc, S.E, 2015.
- [7] Kim Topley, *JavaFX Developer's Guide*, Pearson Education, S.E, 2011.
- [8] Héctor Andrade, Victor Rebolloso, *Guía para el Desarrollo de Aplicaciones Web Responsivas Utilizando el Stack MEAN*, S.E, 2016.
- [9] Tim Bray, Jean Paoli, et. al., *Extensible Markup Language (XML) 1.0*, World Wide Web Consortium, 2008, recuperoado de <https://www.w3.org/TR/REC-xml/> el 19/07/2017.
- [10] Tim Bray, Jean Paoli, et. al., *Extensible Markup Language (XML) 1.0*, World Wide Web Consortium, 2008, recuperoado de <https://www.w3.org/TR/REC-xml/> el 19/07/2017.
- [11] Oracle, *Introduction to FXML*, 2013, recuperado de [https://docs.oracle.com/javase/8/javafx/api/javafx/fxml/doc-files/introduction\\_to\\_fxml.html](https://docs.oracle.com/javase/8/javafx/api/javafx/fxml/doc-files/introduction_to_fxml.html) el 16/05/2017.
- [12] Oracle, *JavaFX CSS Reference Guide*, recuperado de <http://docs.oracle.com/javafx/2/api/javafx/scene/doc-files/cssref.html> el 02/06/2017.
- [13] Gluon, *Web site JavaFXPorts*, recuperado de <http://gluonhq.com/products/mobile/javafxports/> el 20/07/2017.

## **Anexos**

## **Anexo A**

# **Introducción a JavaFX**

Los temas incluidos en este documento son:

1. Introducción a JavaFX
2. Evolución de los GUI Toolkits
3. Ventajas de JavaFX
4. Arquitectura de JavaFX
5. Scene Graph

# 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 Environment. 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 Toolkits 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.

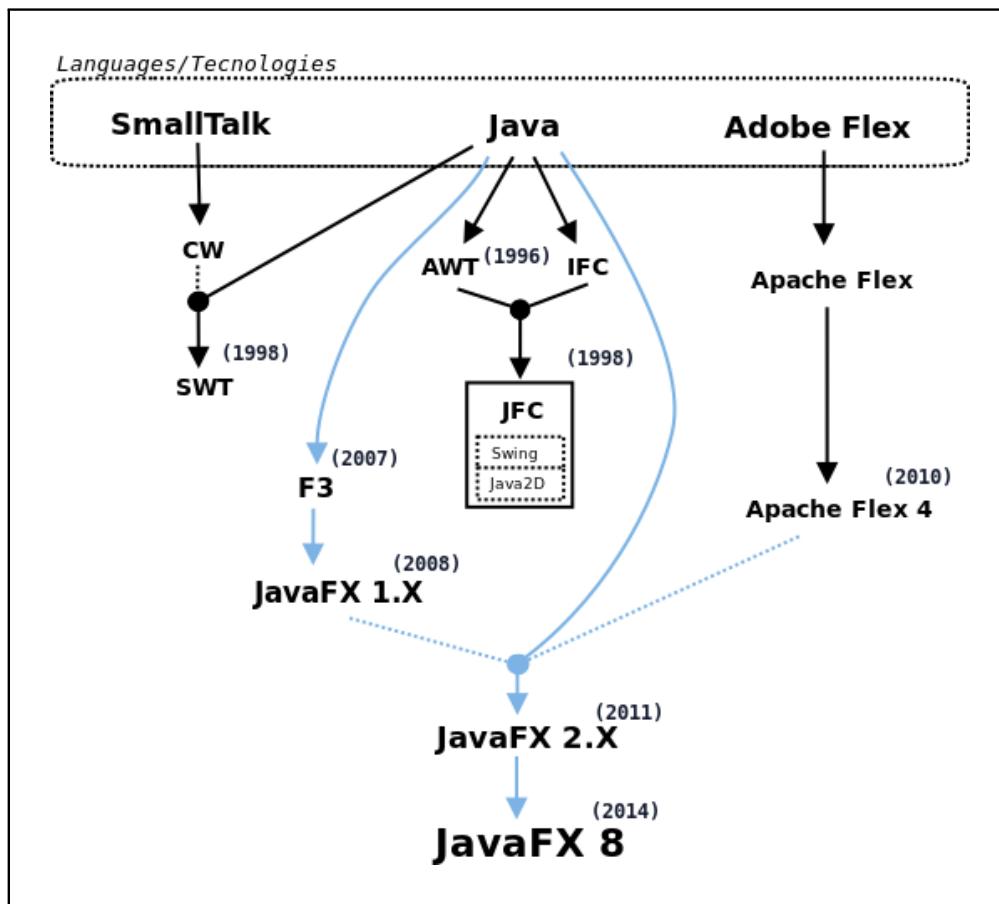


Figura 1.1: Evolución de las herramientas de creación de interfaces gráficas.

### 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.

---

Tópicos Avanzados de Programación

## Capítulo: Introducción

- 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 MXML 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.

---

Tópicos Avanzados de Programación

## Capítulo: Introducción

- 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 beneficio 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.

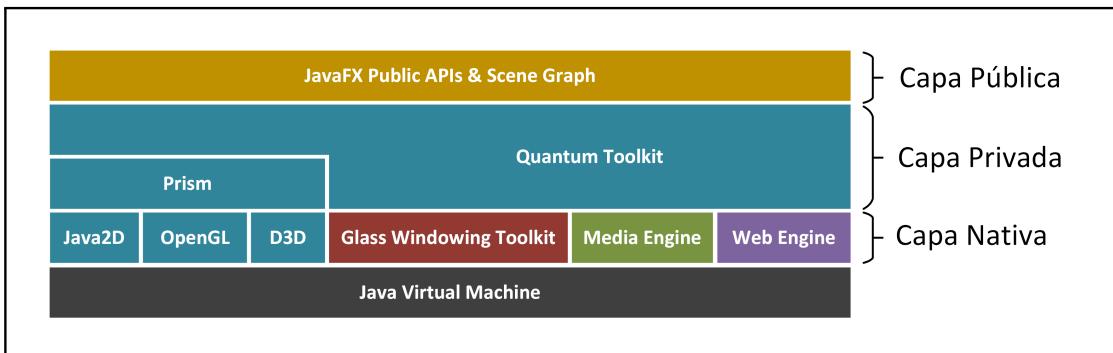


Figura 1.2: Arquitectura de JavaFX

### 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 embeder páginas HTML que hagan uso de CSS y JavaScript dentro de aplicaciones de JavaFX así como videos 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 características opcionales.
javafx.application.ConditionalFeature;

// Método que comprueba si una característica 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 . <sup>a</sup> lto nivel como 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 parent 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 void 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 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-

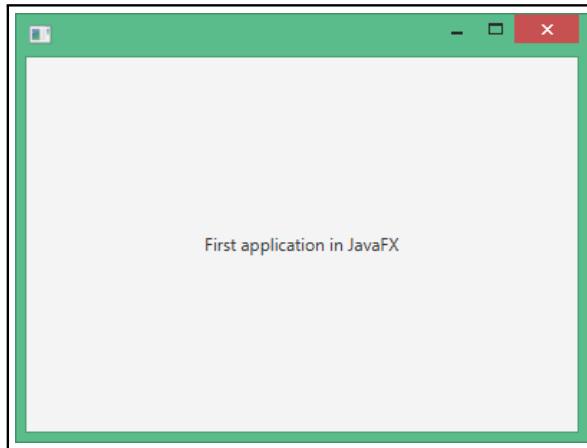


Figura 1.3: Ventana de la primera aplicación

Tabla 1.2: Métodos del ciclo de vida

Método	Descripción
launch(String[])	Método estático invocado desde el método <i>main</i> , 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.

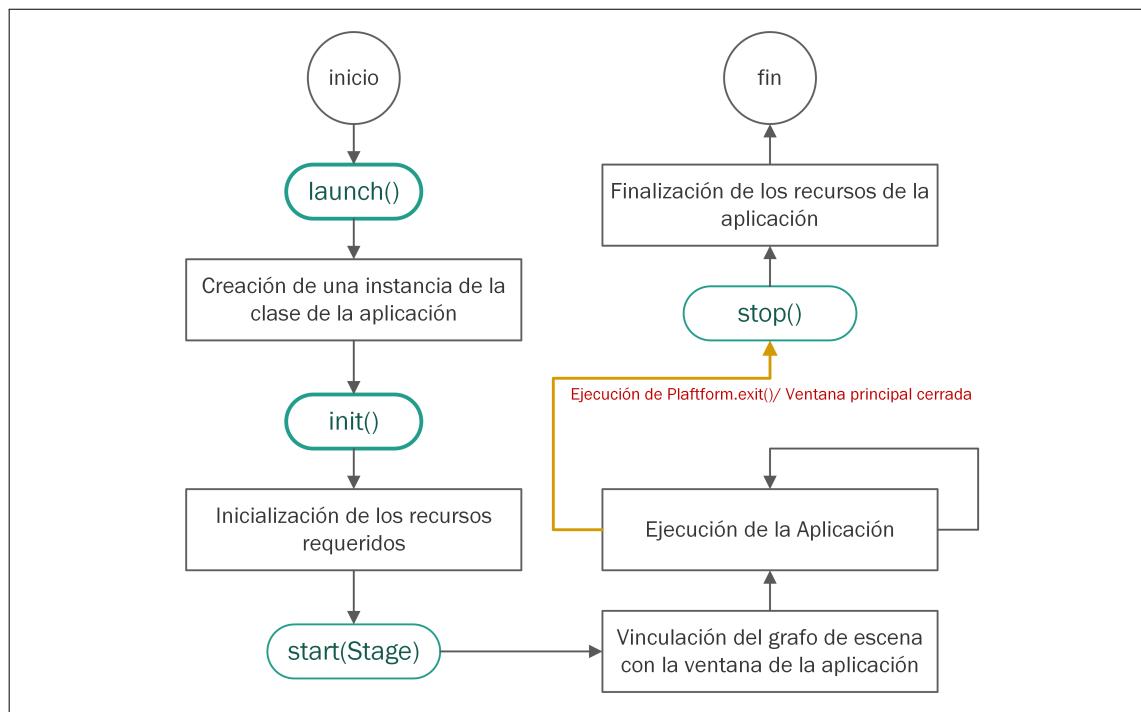


Figura 1.4: Flujo del ciclo de vida de una aplicación JavaFX

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

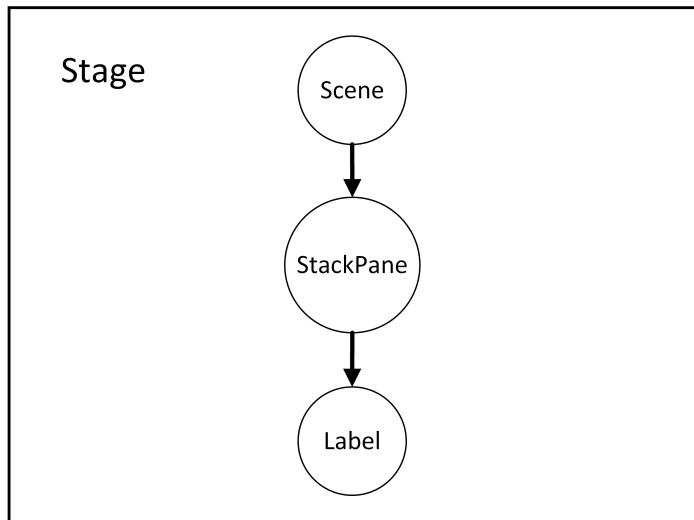


Figura 1.5: Grafo de escena del Código 1

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.

## **Anexo B**

# **Elementos de Interfaces Gráficas**

Los temas incluidos en este documento son:

1. Componentes Gráficos
2. Contenedores Gráficos
3. Administradores de Diseño
4. Diseño de una Interface Gráfica

## Unidad 1

# Elementos de Interfaces Gráficas

### Elementos de una Interfaz Gráfica

Una Interfaz Gráfica de Usuario (GUI, Graphical User Interface) es el conjunto de componentes gráficos que posibilitan la interacción entre el usuario y la aplicación: ventanas, botones, listas, listas desplegables, cajas de diálogo, campos de texto, etc.

Desde el punto de vista de Java, toda interfaz gráfica tiene:

- **Componentes:** Objetos que representan un elemento gráfico (botón, cuadro de texto).
- **Contenedores:** Objetos que contienen componentes (Ventanas, Cuadros de Dialogo).
- **Administradores de diseño:** Objetos que controlan la forma en que se ubican los componentes dentro de un contenedor.
- **Contexto Gráfico:** Área de un componente gráfico donde se pueden dibujar primitivas gráficas (líneas, curvas) y colocar imágenes.
- **Eventos:** Objetos que representan un cambio en un componente, generalmente producido por el usuario al realizar alguna operación.

### Componentes gráficos

Un componente es un objeto que tiene una representación gráfica que puede ser presentada en una pantalla y puede interactuar con el usuario. La clase abstracta Control contiene los métodos comunes para todos los componentes gráficos.

Los desarrolladores cuentan con tres alternativas de herramientas para construir interfaces gráficas en Java. La más antigua, AWT; la más utilizada y con el mayor tiempo en la preferencia de los desarrolladores, SWING; y la más reciente, JavaFX, que tiene cambios importantes en el paradigma de construcción de interfaces.

A continuación se presentan componentes de las tres alternativas.

Tabla 1.1: Métodos más utilizados en un componente gráfico

Método	Descripción
setBackground(Background b)	Define fondo, color o imagen, del componente.
setCursor(Cursor c)	Define el cursor a mostrar sobre el componente.
setBorder(Border b)	Define los bordes del componente.

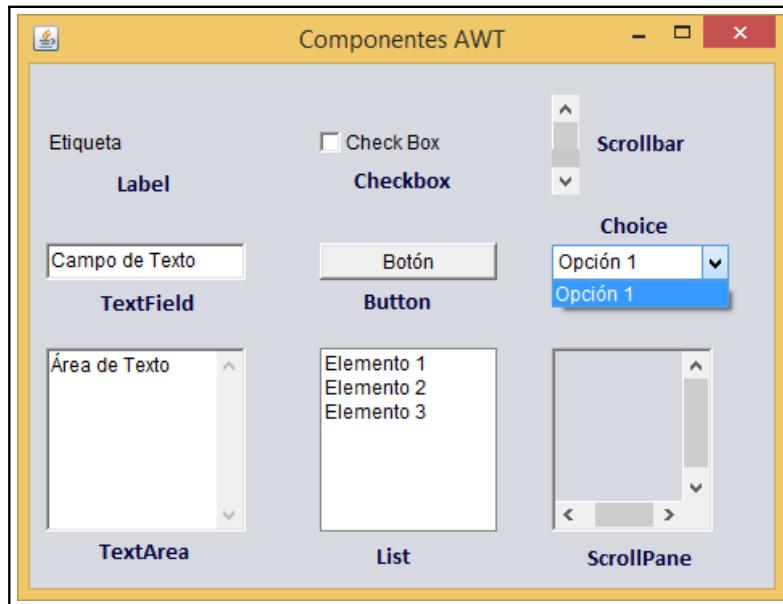


Figura 1.1: Algunos componentes de AWT

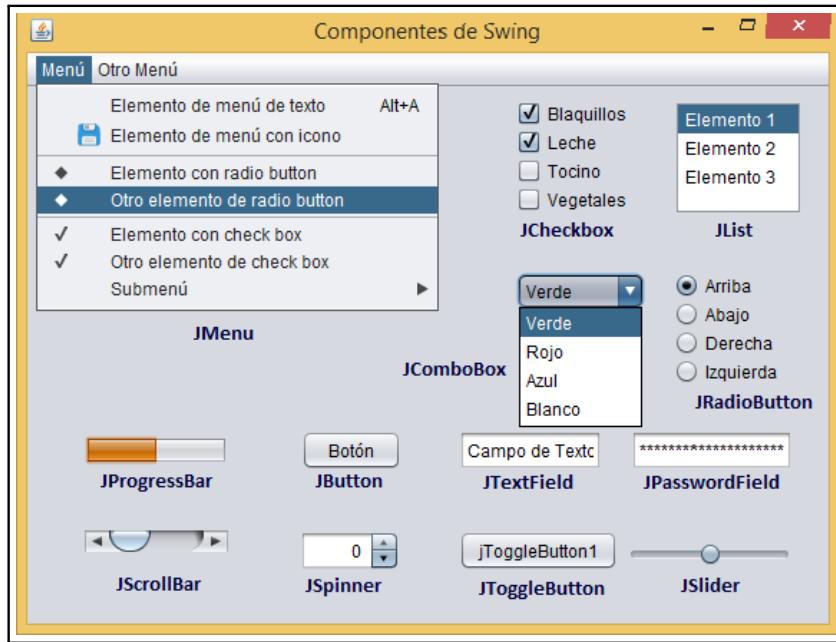


Figura 1.2: Algunos componentes de Swing

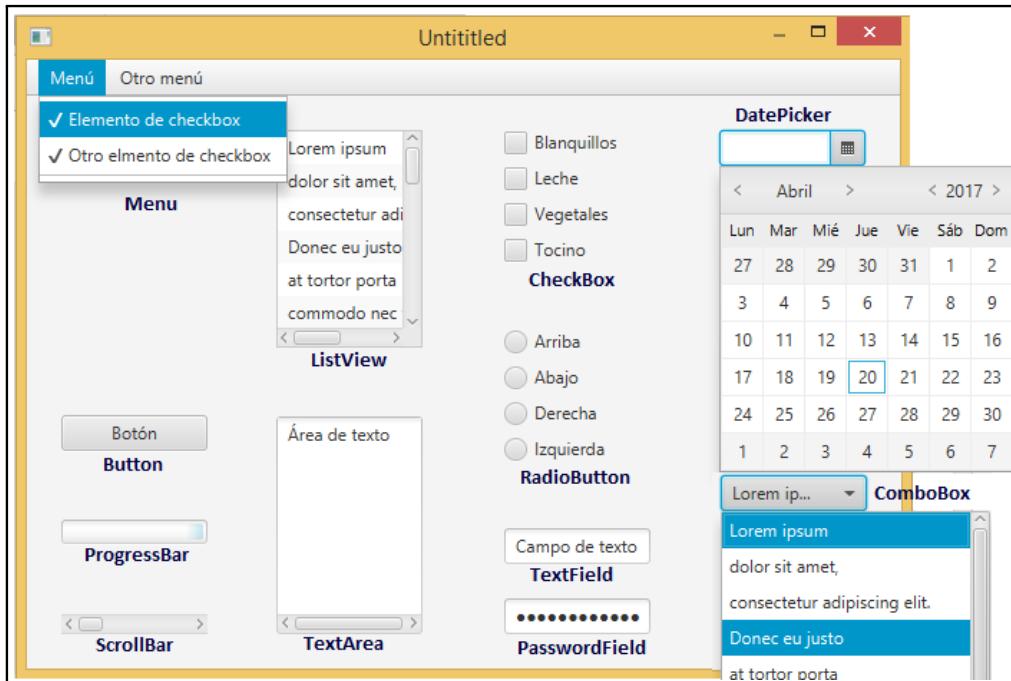


Figura 1.3: Algunos componentes de JavaFX

## Contenedores gráficos

Un contenedor es un componente gráfico que tiene la capacidad de almacenar otros componentes gráficos.

Tabla 1.2: Métodos más utilizados en contenedores

Método	Descripción
setScene(Scene value)	Define la escena que se mostrará en el contenedor.
show()	Hace visible el contenedor.
sizeToScene()	Redimensiona el contenedor al tamaño de su contenido.

## Administradores de diseño

Un administrador de diseño (layout) es un objeto que se encarga de controlar la ubicación y distribución de los componentes dentro de un contenedor. Un layout se considera otro nodo dentro del grafo de escena de una interfaz gráfica, y más que modificar el comportamiento de los contenedores principales son contenedores en sí.

## Diseño de una interfaz gráfica

Una interfaz gráfica común tiene los siguientes elementos:

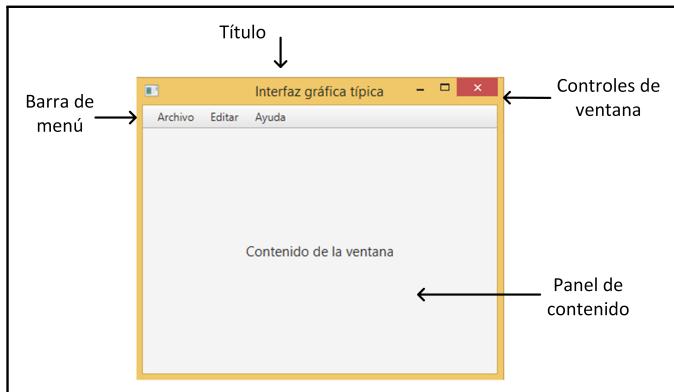


Figura 1.4: Elementos comunes de una interfaz gráfica de usuario

El diseño de una interfaz gráfica debe ser jerárquico. La ventana estará construida por los siguientes elementos:

- Una barra de menú (opcional).
- Un contenedor gráfico para los componentes que definen la interfaz gráfica.
- Un objeto que maneje los eventos de la interfaz.

### Barra de Menú

Es un área de la interfaz de usuario que indica y presenta las opciones o herramientas de una aplicación dispuestas en menús desplegables. En la mayoría de entornos de escritorio los diferentes menús presentes en estas barras pueden ser desplegados por medio de atajos de teclado, al mantener presionada la tecla ALT y la tecla correspondiente a la letra subrayada en la barra de menú.

### Contenedor gráfico

Define el área donde se construirá la interfaz gráfica. Este contenedor sirve como organizador de los componentes gráficos.

### Manejador de Eventos

Cuando un usuario utiliza una interfaz gráfica, realiza cambios sobre los componentes (pulsa un botón, captura sobre un campo, etc). Estos cambios se reflejan en objetos denominados eventos.

Los eventos deben ser atendidos para que la interfaz tenga el comportamiento esperado por el usuario, por lo cual toda interfaz gráfica debe utilizar un elemento manejador de eventos.

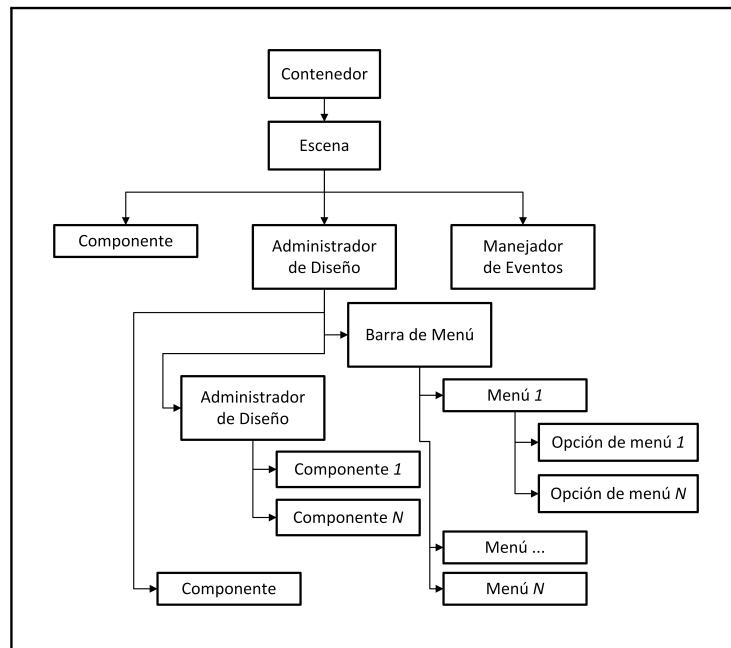


Figura 1.5: Estructura general de una interfaz gráfica de usuario

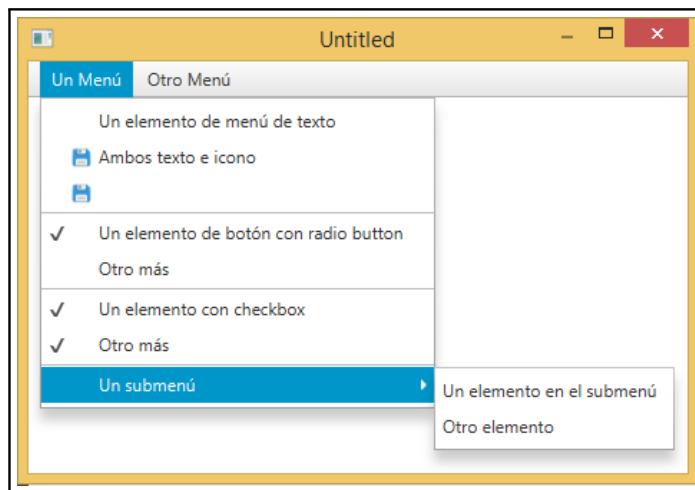


Figura 1.6: Ejemplo de una barra de menú y varias opciones de menú

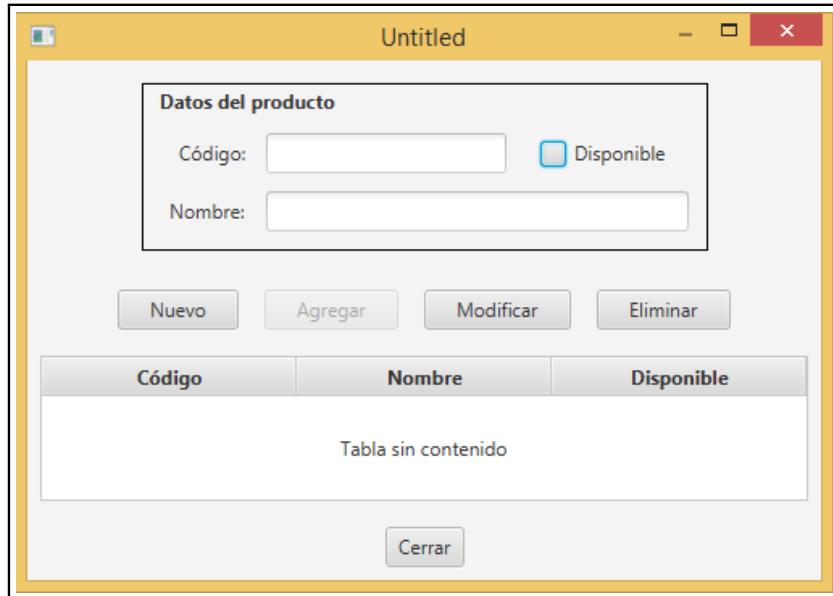


Figura 1.7: Ejemplo de un contenedor con varios componentes gráficos

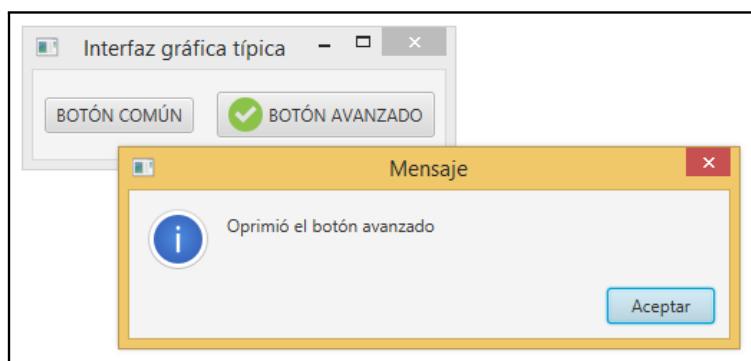


Figura 1.8: Ejemplo de un manejo de evento. Al oprimir el botón, se despliega una ventana de diálogo

### Estructura de clases para una interfaz gráfica

En este curso, cada aplicación gráfica estará diseñada dentro de un paquete, y tendrá una clase aplicación (donde se define el método main), y una o varias clases instanciables que definen los menús, los contenedores y los demás elementos necesarios para la interfaz.

La clase aplicación creará una instancia del **contenedor principal** (la ventana), definirá su configuración (tamaño, ubicación), le asignará la barra de menú correspondiente (si esta existe) y el contenedor para los componentes gráficos. Esta clase también creará los objetos que atenderán los eventos producidos en la aplicación para el comportamiento adecuado de esta.

Código 1.1: Creación de una ventana simple

```
import javafx.application.Application;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;

public class Ventana extends Application {

    @Override
    public void start(Stage primaryStage) {
        Parent root = new Pane();
        Scene scene = new Scene(root);

        primaryStage.setTitle("Ejemplo_de_Ventana");
        primaryStage.setScene(scene);
        primaryStage.setWidth(400);
        primaryStage.setHeight(300);
        primaryStage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

## **Anexo C**

# **Librerías de Interfaces Gráficas**

Los temas incluidos en este documento son:

1. Componentes Genéricos - AWT
2. Contenedores Especializados - Swing
3. Componentes de JavaFX
4. Stage, Scene y Layouts
5. Scene Builder y FXML

# Unidad 1

## Elementos de Interfaces Gráficas

### Librerías de Interfaz Gráfica

Java posee varias alternativas para la creación de interfaces gráficas para sus aplicaciones.

AWT: El Abstract Window Toolkit es el paquete básico de construcción de interfaces gráficas de Java.

Swing: Se incluyó dentro de JFC en la versión 1.2 de Java con el objetivo de que los componentes de la interface fueran independientes del sistema operativo en que se ejecutara la aplicación.

Java2D: Modela primitivas gráficas (líneas, elipses, curvas) en objetos y permite aplicarles transformaciones (rotación, escalado).

Java3D: Utilizado para crear ambientes y figuras tridimensionales.

JavaFX: Es la propuesta más reciente que integra varias mejoras al sistema de construcción de interfaces gráficas.

### Componentes Genéricos (AWT)

AWT es el acrónimo del Abstract Window Toolkit para Java. Se trata de una biblioteca de clases Java para el desarrollo de Interfaces Gráficas de Usuario. La versión original de AWT se desarrolló en sólo dos meses y es la parte más débil de todo lo que representa Java como lenguaje. AWT contiene:

- Un gran conjunto de componentes de interfaz de usuario.
- Un robusto modelo de manejo de eventos.
- Herramientas gráficas y de imagen, incluyendo forma, color y tipo de letra.
- Administradores de diseño (layout), para un manejo de ventanas flexible que no dependan de una tamaño o resolución específico.
- Clases de transferencia de datos, para copiar y pegar a través del portapapeles de la plataforma en donde se ejecuta la aplicación.

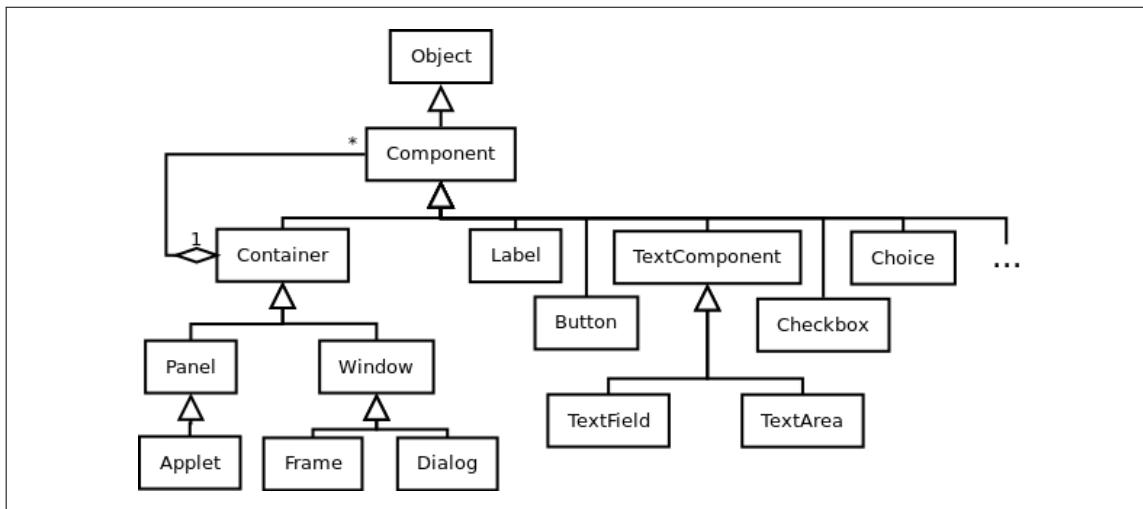


Figura 1.1: Diagrama de clases de los componentes de AWT

## Componentes Especializados (Swing)

- El paquete Swing es parte de la JFC (Java Foundation Classes) en la plataforma Java.
- La JFC provee facilidades para ayudar a los desarrolladores a construir interfaces gráficas.
- Swing abarca componentes como botones, tablas, marcos, etc.
- Las componentes Swing se identifican porque pertenecen al paquete `javax.swing`.

Antes de la existencia de Swing, las interfaces gráficas de usuario se construían con AWT (Abstract Window Toolkit), de quien Swing hereda todo el manejo de eventos. Usualmente, para toda componente AWT existe una componente Swing que la reemplaza, por ejemplo, la clase `Button` de AWT es reemplazada por la clase `JButton` de Swing (el nombre de todas las componentes Swing comienza con "J").

Las componentes de Swing utilizan la infraestructura de AWT, incluyendo el modelo de eventos AWT, el cual rige cómo una componente reacciona a eventos tales como, eventos de teclado, mouse, etc.

## Componentes de JavaFX

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 beneficio 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

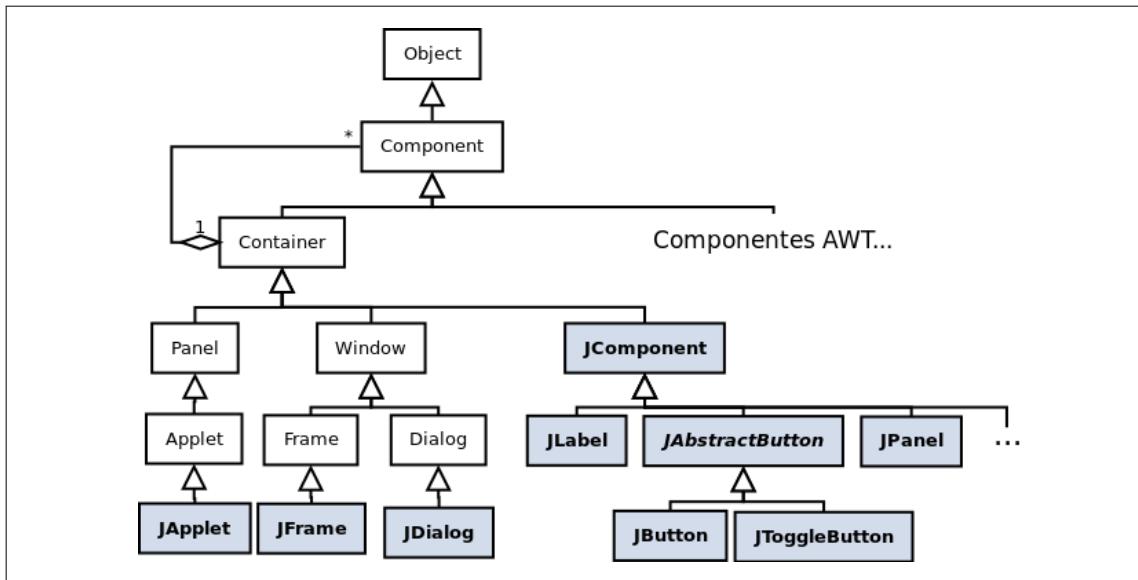


Figura 1.2: Diagrama de clases de los componentes de SWING

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.

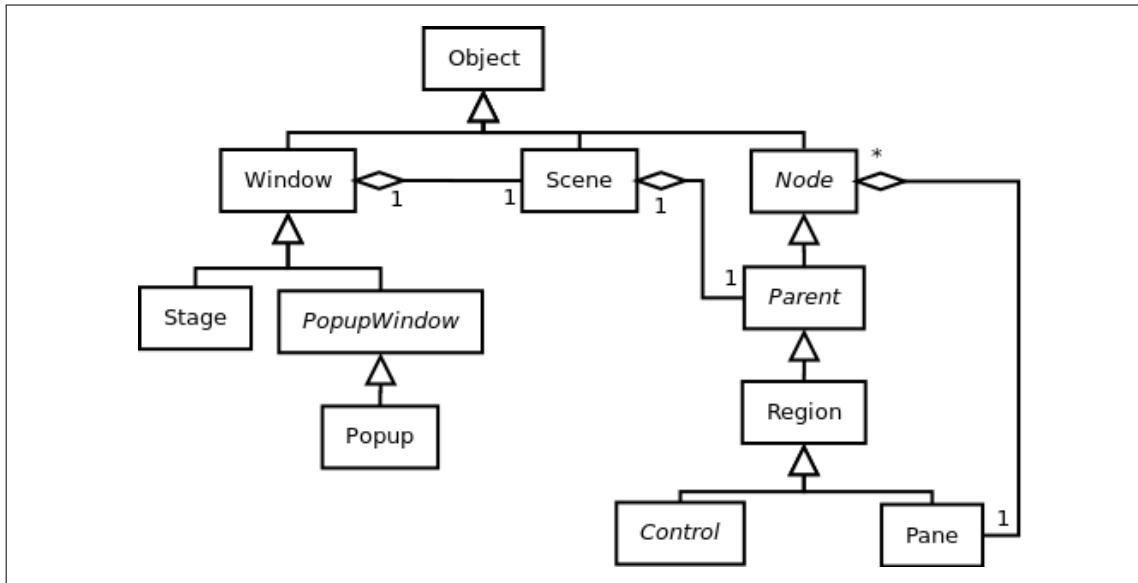


Figura 1.3: Diagrama de clases de los componentes de JavaFX

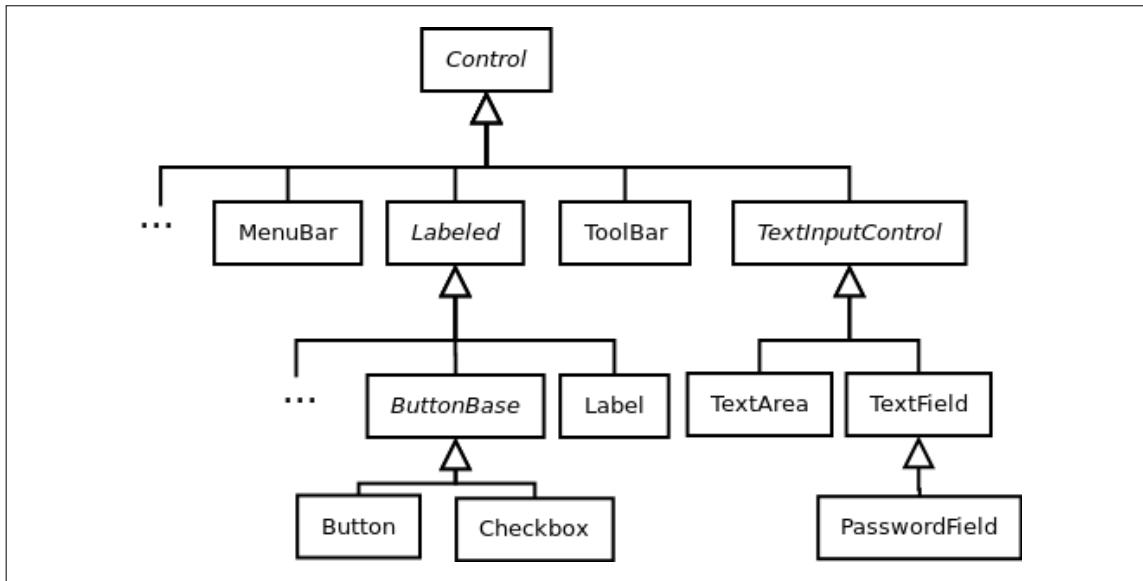


Figura 1.4: Diagrama de clases de algunos controles de JavaFX

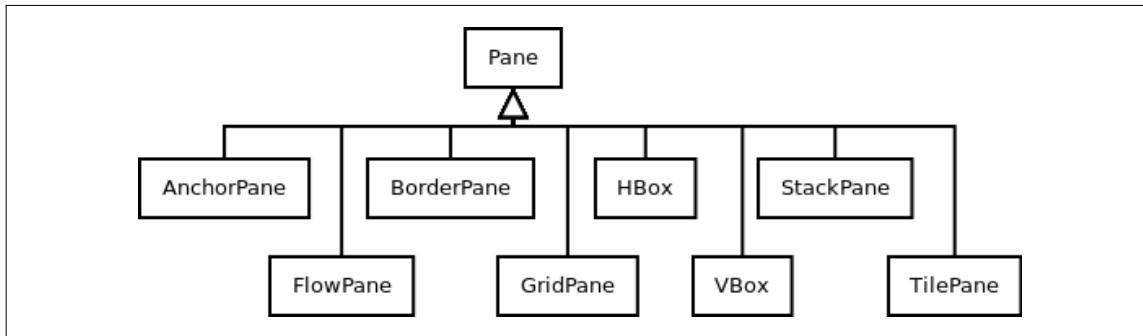


Figura 1.5: Diagrama de clases de los contenedores/administradores de diseño de JavaFX

Tabla 1.1: Métodos comunes de la clase Stage

Método	Descripción
setScene(Scene value)	Añade el grafo de escena al contenedor.
setTitle(String title)	Establece el título para mostrar en la barra superior del contenedor.
show()	Vuelve visible para el usuario el contenedor.
initModality(Modality mod)	Establece la modalidad en que va a funcionar el contenedor. Las modalidades son: Modality.NONE, Modality.WINDOW_MODAL, Modality.APPLICATION_MODAL.

## Stage

La clase Stage modela el contenedor de mayor nivel en las interfaces construidas con JavaFX. Al iniciar una aplicación JavaFX un objeto Stage, comúnmente llamado *primaryStage*, es inyectado al método *start* del ciclo de vida de la aplicación, ese objeto será el principal contenedor de la aplicación y en él se mostrará el grafo de escena de la interface.

Adicionalmente se pueden crear más objetos Stage durante la ejecución de la aplicación.

## Scene

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 asociado a ella. El grafo contiene nodos que 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 parent 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.

## Paneles y Administradores de Diseño

Los paneles son contenedores que agrupan a otros componentes. Anteriormente, con Swing, los conceptos de contenedor y administrador de diseño eran independientes, ahora con JavaFX ambos conceptos están tan ligados al grado que las clases contenedoras ya tienen un administrador de diseño establecido que no puede ser cambiado.

Como ya se explicó en el tema Elementos de una interfaz Gráfica, la importancia de los administradores de diseño reside, principalmente, en la capacidad para ubicar los componentes gráficos de la interface siguiendo un patrón de distribución y ubicación. JavaFX ofrece 9 administradores de diseño distintos, suficientes para la mayoría de los casos. Las clases están contenidas en el paquete `javafx.scene.layout`.

A continuación, la tabla 1.5, describe brevemente cada administrador de diseño.

Tabla 1.2: Administradores de diseño en JavaFX

Layout	Descripción
AnchorPane	Coloca los componentes en una posición relativa al los bordes del contenedor.
BorderPane	Divide el administrador de diseño en 5 secciones: norte, sur, este, oeste y centro.
FlowPane	Ubica los componentes que contiene en posición superior izquierda, uno después de otro conforme estos fueron añadidos. Ajusta el tamaño de sus componentes al mínimo.
GridPane	Divide el contenedor en filas y columnas permitiendo ubicar los controles en celdas específicas.
HBox	Coloca los controles siguiendo un acomodo horizontal ordenando de izquierda a derecha.
Pane	Clase base de la que derivan el resto de administradores de diseño. Útil cuando se ubican los componentes de la interfaz mediante posición absoluta.
StackPane	Ubica los componentes gráficos al centro del panel apilandolos uno encima de otro acorde al orden en que fueron añadidos.
TilePane	Ubica los componentes en una malla horizontal o vertical cuyas celdas poseen el mismo espacio disponible respecto al tamaño mínimo preferido del componente más grande contenido.
VBox	Coloca los controles verticalmente ordenando de arriba hacia abajo.

Código 1.1: Aplicación de demostración con componentes y administrador de diseño sencillo.

```

import javafx.application.Application;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;

public class Main extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception{
        Parent root = new Panel();
        primaryStage.setTitle("Hello World");
        primaryStage.setScene(new Scene(root, 600, 500));
        primaryStage.show();
    }

    public static void main(String[] args) { launch(args); }
}

```

Código 1.2: Clase auxiliar para definir la interfaz de usuario.

```

import javafx.scene.control.*;
import javafx.scene.layout.Pane;

public class Panel extends Pane {

    public Panel() { build(); }

    private void build() {
        //BOTÓN
        Button btn = new Button("Mi botón");
        locateControl(btn, 50, 50);
        //ETIQUETA
        Label lbl = new Label("Este es un mensaje:");
        locateControl(lbl, 300, 50);
        //CAMPO DE TEXTO
        TextField input = new TextField();
        input.setPromptText("Valor de entrada:");
        locateControl(input, 50, 120);
        //ÁREA DE TEXTO
        TextArea area = new TextArea("Captura de texto en varias líneas");
        area.setPrefColumnCount(15);
        area.setPrefRowCount(3);
        locateControl(area, 300, 120);
        //CASILLAS DE VERIFICACIÓN
        CheckBox check1 = new CheckBox("Uno"), check2 = new CheckBox("Dos");
        check1.setSelected(true);
        locateControl(check1, 50, 260);
        locateControl(check2, 50, 290);
        //GRUPO DE BOTONES
        ToggleGroup group = new ToggleGroup();
        RadioButton rBtn1 = new RadioButton("Rojo"), rBtn2 = new RadioButton("Verde"),
            rBtn3 = new RadioButton("Azul");
        rBtn1.setToggleGroup(group);
        rBtn2.setToggleGroup(group);
        rBtn3.setToggleGroup(group);
        locateControl(rBtn1, 300, 260);
        locateControl(rBtn2, 370, 260);
        locateControl(rBtn3, 450, 260);
        //LISTA DESPLEGABLE
        ComboBox combo = new ComboBox();
        combo.getItems().addAll("Cyan", "Magenta", "Amarillo", "Negro");
        combo.setValue("Cyan");
        locateControl(combo, 50, 340);
        //LISTA DE ELEMENTOS
        ListView<String> list = new ListView<>();
        list.getItems().addAll("Opción_1", "Opción_2", "Opción_3", "Opción_4", "Opción_5");
        list.setPrefHeight(100);
        locateControl(list, 300, 340);
    }

    private void locateControl(Control c, double x, double y) {
        c.setLayoutX(x); c.setLayoutY(y); getChildren().add(c);
    }
}

```

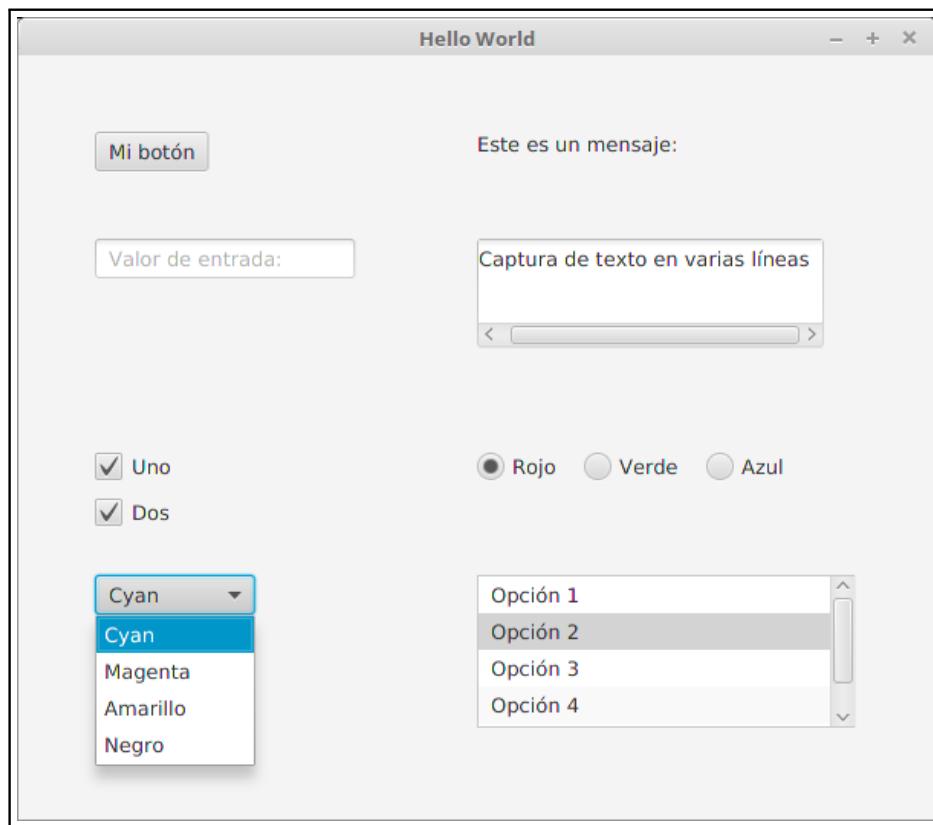


Figura 1.6: Aplicación producida los códigos 1.1 y 1.2

Tabla 1.3: Métodos asociados al manejo de menús

Método	Descripción
getMenus()	Método de la clase MenuBar que devuelve la lista de menús asociados a la barra de menú.
getItems()	Método de la clase Menu que devuelve la lista de elementos de menú asociados.

## Menús

En JavaFX los menús pueden añadirse a cualquier administrador de diseños a diferencia de Swing donde solo se podían añadir a instancias de JFrame. Se requieren tres clases para hacer uso de menús en una interfaz gráfica.

- **MenuBar:** Define la barra de menú para la ventana.
- **Menu:** Define el menú incluido en la barra menú.
- **MenuItem:** Define la opción de un menú.

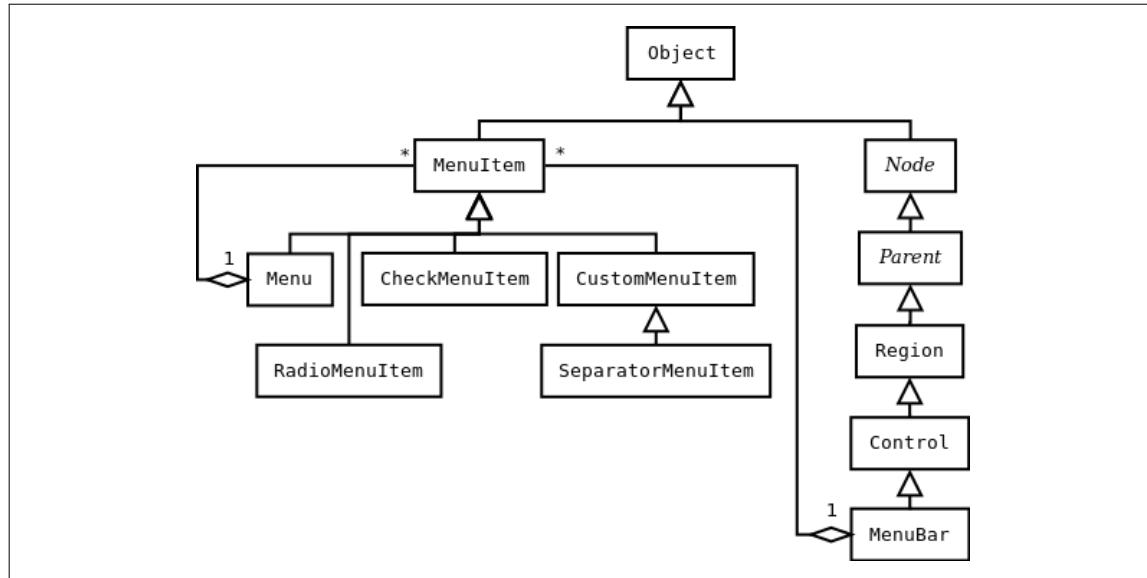


Figura 1.7: Diagrama de clases de los componentes de tipo Menu

Código 1.3: Aplicación de demostración con componentes de tipo menú.

```

import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.layout.BorderPane;
import javafx.stage.Stage;

public class Main extends Application {

    @Override
    public void start(Stage primaryStage) {
        BorderPane root = new BorderPane();
        MyMenuBar menuBar = new MyMenuBar();
        root.setTop(menuBar);
        primaryStage.setTitle("Ejemplo con menús");
        primaryStage.setScene(new Scene(root, 600, 500));
        primaryStage.show();
    }

    public static void main(String[] args) { launch(args); }
}

```

Código 1.4: Clase auxiliar para construir la barra de menú de la aplicación.

```

import javafx.scene.control.Menu;
import javafx.scene.control.MenuItem;
import javafx.scene.control.SeparatorMenuItem;

public class MyMenuBar extends javafx.scene.control.MenuBar {

    public MyMenuBar() { build(); }

    private void build() {
        Menu file = new Menu("Archivo");
        addMenu(file);
        addMenu(new Menu("Editar"));
        addMenu(new Menu("Ayuda"));

        addMenuItems(file, new MenuItem("Abrir"), new MenuItem("Guardar"),
                    new MenuItem("Cerrar"), new SeparatorMenuItem());

        Menu props = new Menu("Propiedades");
        addMenuItems(props, new MenuItem("Propiedad_1"), new MenuItem("Propiedad_2"),
                    new MenuItem("Propiedad_3"));

        addMenuItems(file, props, new SeparatorMenuItem(), new MenuItem("Salir"));
    }

    private void addMenu(Menu mi) { getMenus().add(mi); }

    private void addMenuItems(Menu m, MenuItem... mis) { m.getItems().addAll(mis); }
}

```

Utilizando el código 1.3 junto al código 1.5 se obtiene una aplicación que ejemplifica el uso de caracteres mnemónicos para identificar los menús al presionar la tecla ALT y el uso de combinaciones de teclas para seleccionar un item de menú sin la necesidad de interactuar directamente con la interface.

#### KeyCombination

Como su nombre lo indica, KeyCombination representa una combinación de teclas que puede

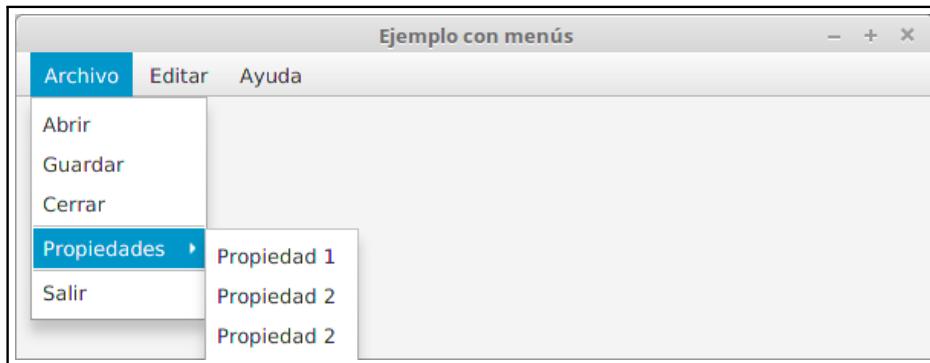


Figura 1.8: Aplicación producida con los códigos 1.3 y 1.4

Tabla 1.4: Métodos de utilidad de MenuItem

Método	Descripción
setMnemonicParsing(boolean b)	Si la propiedad mnemonicParsing es establecida como <code>true</code> se hará un parseo del texto del MenuItem en busca del carácter <code>_</code> , al ser hallado se establecerá el carácter que le suceda como mnemónico.
setAccelerator(KeyCombination k)	Define la combinación de teclas que ejecutarán la acción asociada a un MenuItem.
setGraphic(Node g)	Establece un componente gráfico para mostrar con junto al MenuItem.

asociarse a una opción de menú para ejecutar su acción. Se tienen dos alternativas para la creación de instancias de KeyCombination: con alguno de sus dos constructores y con un método estático `keyCombination(String s)`.

Código 1.5: Aplicación de demostración con acciones en menús e imágenes.

```

public class MyMenuBar extends MenuBar{
    public MyMenuBar() { build(); }

    private void build() {
        //El guión bajo indica cual será el carácter mnemónico del menú.
        Menu file = new Menu("_Archivo");
        Menu edit = new Menu("_Editar");

        enableMnemonic(file);
        enableMnemonic(edit);

        addMenus(file, edit, new Menu("Ayuda"));

        addMenuItems(file,
            new MenuItem("Abrir") , new MenuItem("Guardar"),
            new MenuItem("Cerrar"), new SeparatorMenuItem());

        Menu props = new Menu("Propiedades");
        addMenuItems(props,
            new MenuItem("Propiedad_1"), new MenuItem("Propiedad_2"),
            new MenuItem("Propiedad_2"));

        addMenuItems(file,
            props, new SeparatorMenuItem(), new MenuItem("Salir"));

        MenuItem copy = new MenuItem("Copiar");

        //Asociación de la combinación de teclas con un item de menú.
        copy.setAccelerator(KeyCombination.keyCombination("SHORTCUT+C"));
        copy.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent event) {
                System.out.println("Something was copied");
            }
        });

        InputStream stream = getClass().getResourceAsStream("copy-icon.png");
        copy.setGraphic(new ImageView(new Image(stream)));
        addMenuItems(edit, copy);
    }
}

```

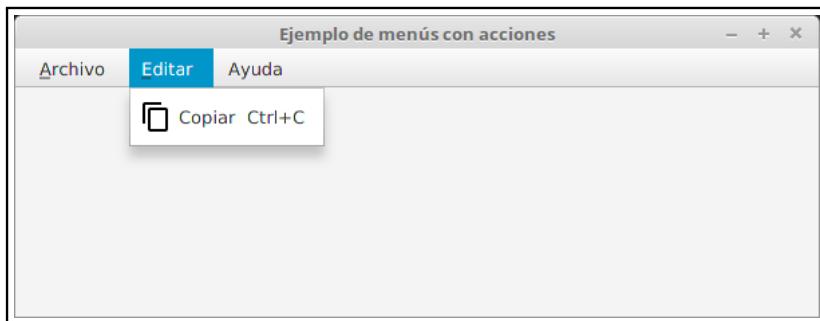


Figura 1.9: Aplicación producida con los códigos 1.3 y 1.5

## TabPane

TabPane es un control de interfaz gráfica, no un administrador de diseño como se podría pensar. La ventaja de este control es que funciona como un contenedor en el que se pueden organizar secciones de la interface en pestañas.

Estas son algunas características de TabPane:

- Solamente es visible el contenido de una de las pestañas a la vez.
- Las pestañas pueden incluir texto y/o un ícono.
- La ubicación de las pestañas puede ser en la parte superior, inferior, derecha o izquierda del contenedor.

Tabla 1.5: Recursos útiles para el uso de TabPane

Clase, Constructor o Método	Descripción
javafx.scene.control.Tab	Clase modelo de las pestañas.
TabPane(Tab... tabs)	Construye un nuevo TabPane con las pestañas especificadas en el constructor.
getTabs()	Devuelve la lista de pestañas del contenedor, a esa lista se pueden añadir o quitar Tabs.
setSide(Side s)	Define la ubicación de las pestañas.

Los códigos 1.6, 1.7 y 1.8 ejemplifican una aplicación que utiliza el control TabPane. Como extra también se demuestran las capacidades programáticas para configurar apariencia, ubicación, tamaño y tamaño referente al contenedor de los componentes.

Código 1.6: Aplicación de demostración del uso de TabPane.

```
public class Main extends Application {
    @Override
    public void start(Stage primaryStage) {
        TabPane root = new MyTabPane();
        primaryStage.setTitle("Ejemplo_de_TabPane_y_Paneles_Complejos");
        primaryStage.setScene(new Scene(root, 600, 400));
        primaryStage.show();
    }
    public static void main(String[] args) { launch(args); }
}
```

Código 1.7: Subclase de TabPane para la demostración (Parte 1).

```

public class MyTabPane extends TabPane {

    public MyTabPane() { build(); }

    private void build() {
        getTabs().add(createGridPane());
        getTabs().add(createSimpleGridPane());
        getTabs().add(createBorderPane());
        getTabs().add(createMixedPane());
    }

    private Tab createGridPane() {
        Tab tab = new Tab("GridPane con Elementos Autoajustables");
        GridPane content = new GridPane();
        Pane red = new Pane(), green = new Pane(), blue = new Pane();

        //Asignación de colores de background con propiedades CSS
        red.setStyle("-fx-background-color:#FF0000");
        green.setStyle("-fx-background-color:#00FF00");
        blue.setStyle("-fx-background-color:#0000FF");

        //Configuración del ajustado automático
        setAutogrow(red); setAutogrow(green); setAutogrow(blue);
        content.add(red, 0, 0);
        content.add(green, 1, 0);
        content.add(blue, 0, 1);

        tab.setContent(content);
        return tab;
    }

    private Tab createSimpleGridPane() {
        Tab tab = new Tab("GridPane con Elementos Alineados");
        GridPane content = new GridPane();
        Button left = new Button("Izquierda"), center = new Button("Centro"),
            rightTop = new Button("Derecha"), leftBottom = new Button("Izquierda Abajo");

        //Distribución equitativa del ancho disponible en ambas columnas
        ColumnConstraints cc = new ColumnConstraints();
        cc.setPercentWidth(50);
        content.getColumnConstraints().addAll(cc, cc);

        //Distribución equitativa del alto disponible en ambas filas
        RowConstraints rc = new RowConstraints();
        rc.setPercentHeight(50);
        content.getRowConstraints().addAll(rc, rc);

        //Alineamiento de los botones
        GridPane.setAlignment(center, HPos.CENTER);
        GridPane.setAlignment(rightTop, HPos.RIGHT);
        GridPane.setAlignment(leftBottom, HPos.LEFT);
        GridPane.setAlignment(leftBottom, VPos.BOTTOM);
        content.add(left, 0, 0);
        content.add(center, 1, 0);
        content.add(rightTop, 0, 1);
        content.add(leftBottom, 1, 1);

        tab.setContent(content);
        return tab;
    }

    ...
}

```

Código 1.8: Subclase de TabPane para la demostración (Parte 2).

```

...
private Tab createBorderPane() {
    Tab tab = new Tab("BorderPane_con_Botones_de_Tamaño_Autoajustable");
    BorderPane content = new BorderPane();
    Button top = new Button("Norteño"), bottom = new Button("Sureño"),
        center = new Button("Neutral"), left = new Button("Comunista"),
        right = new Button("Capitalista");

    //Configuración del ajustado automático mediante las medidas máximas de los botones
    top.setMaxWidth(Double.MAX_VALUE);
    bottom.setMaxWidth(Double.MAX_VALUE);
    center.setMaxWidth(Double.MAX_VALUE);
    center.setMaxHeight(Double.MAX_VALUE);
    left.setMaxHeight(Double.MAX_VALUE);
    right.setMaxHeight(Double.MAX_VALUE);

    content.setTop(top);
    content.setCenter(center);
    content.setBottom(bottom);
    content.setLeft(left);
    content.setRight(right);

    tab.setContent(content);
    return tab;
}

private Tab createMixedPane() {
    Tab tab = new Tab("Mezcla_de_Paneles");
    BorderPane content = new BorderPane();
    HBox top = new HBox();

    content.setStyle("-fx-border-color: green; -fx-border-width: 3px");
    top.setAlignment(Pos.CENTER); //← Alineamiento de los elementos
    top.setSpacing(4); //← Asignación de espacio entre elementos
    top.setPadding(new Insets(8)); //← Asignación de un espacio entre los bordes y el
        // contenido

    //Asignación de color de fondo mediante método
    top.setBackground(new Background(new BackgroundFill(Color.CYAN, null, null)));
    top.getChildren().addAll(new Button("Aceptar"), new Button("Cancelar"));
    content.setTop(top);

    tab.setContent(content);
    return tab;
}

private void setAutogrow(Node node) {
    GridPane.setVgrow(node, Priority.ALWAYS);
    GridPane.setHgrow(node, Priority.ALWAYS);
}
}

```

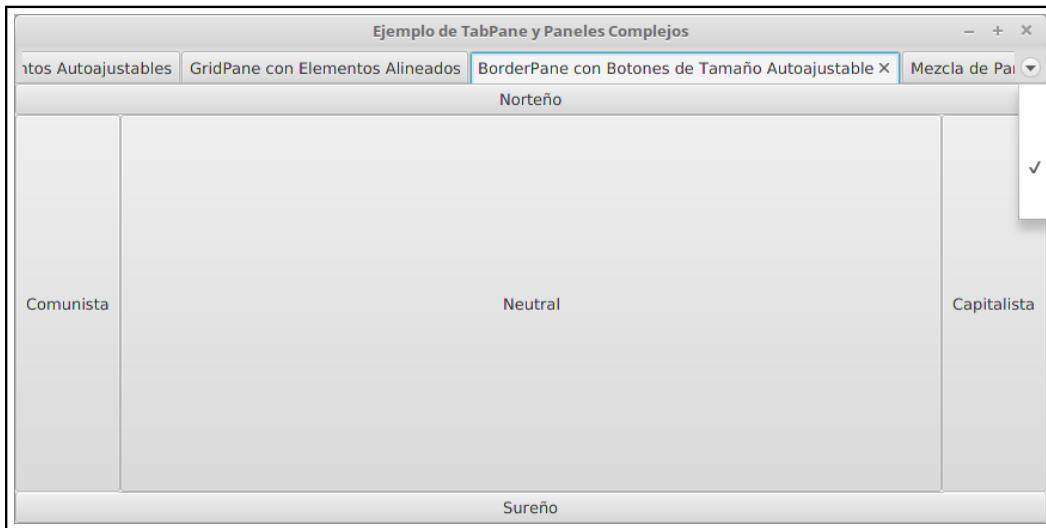


Figura 1.10: Aplicación creada con los códigos 1.6, 1.7 y 1.8

## Dialog: Alert, TextInputDialog y ChoiceDialog

JavaFX provee estas tres clases para mostrar mensajes, pedir un dato o solicitar la selección de una opción al usuario mediante una ventana emergente. La mayoría de los casos en que se pueda necesitar la interacción con el usuario mediante ventanas emergentes se pueden cubrir con estas clases.

Algunas características de las ventanas emergentes son las siguientes:

- Su comportamiento puede ser modal o bloqueante. El comportamiento modal permite al usuario seguir interactuando con la interface de la aplicación mientras que el bloqueante impide cualquier acción que no sea dentro de la ventana emergente.
- Las ventanas emergentes están vinculadas a la ventana principal de la aplicación, si esta se cierra, maximiza o minimiza las ventanas emergentes ejecutarán el mismo comportamiento.
- Son subclases de `javafx.scene.control.Dialog`.

## FileChooser

Permite navegar por el sistema de archivos de la plataforma donde se ejecuta la aplicación y su look and feel (apariencia) es dependiente del sistema operativo. A diferencia de otros controles de interface, la clase `FileChooser` pertenece al paquete `javafx.scene`.

Estos son algunos métodos importantes de esta clase.

Tabla 1.6: Métodos de utilidad de la clase Dialog

Método	Descripción
setTitle(String t)	Define el título que se mostrará en el marco.
setHeaderText(String t)	Define el texto de la cabecera del mensaje.
setContentText(String t)	Define el mensaje de la ventana emergente.
setAlertType(AlertType t)	Establece el tipo de la ventana emergente (En el caso de Alert). Los tipos disponibles son NONE, CONFIRMATION, INFORMATION, WARNING, ERROR.
show()	Muestra la ventana emergente en modo modal (no espera por la interacción del usuario).
showAndWait()	Muestra la ventana emergente en modo bloqueante (espera por la interacción del usuario).

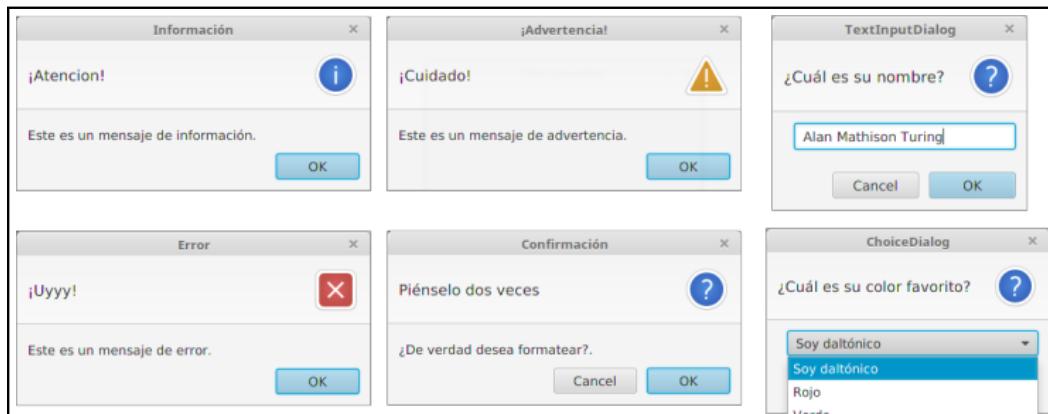


Figura 1.11: Diálogos de ejemplo

Tabla 1.7: Métodos de utilidad de la clase Dialog

Método	Descripción
showOpenDialog(Window w)	Permite seleccionar <b>un</b> archivo mediante el dialogo de navegación del sistema de archivos. Devuelve una instancia de la clase File o null si no se seleccionó un archivo. El parámetro w debe ser la ventana a la que estará vinculada la ventana de navegación.
showOpenMultipleDialog(Window w)	Permite seleccionar <b>múltiples</b> archivos mediante el dialogo de navegación del sistema de archivos. Devuelve una instancia de la clase File o null si no se seleccionó un archivo. El parámetro w debe ser la ventana a la que estará vinculada la ventana de navegación.
showSaveDialog(Window w)	Permite guardar un archivo mediante el dialogo de navegación del sistema de archivos. Devuelve una instancia de la clase File o null si no se seleccionó un archivo. El parámetro w debe ser la ventana a la que estará vinculada la ventana de navegación.
getExtensionFilters()	Devuelve la lista de filtros de archivos por extensión, a esa lista se pueden añadir o quitar filtros.

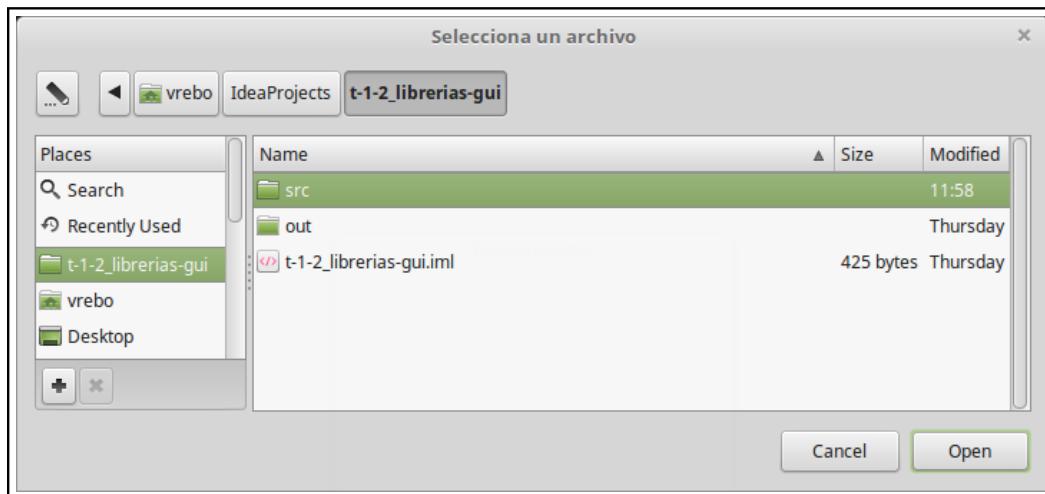


Figura 1.12: Diálogo de navegación por el sistema de archivos

Tabla 1.8: Métodos de utilidad de la clase ScrollPane

Método	Descripción
setHbarPolicy(ScrollBarPolicy p)	Define el comportamiento para mostrar las barra de desplazamiento horizontal. Los valores posibles son <code>ScrollBarPolicy.ALWAYS</code> , <code>ScrollBarPolicy.AS_NEEDED</code> , <code>ScrollBarPolicy.NEVER</code> .
setVbarPolicy(ScrollBarPolicy p)	Define el comportamiento para mostrar las barras de desplazamiento vertical. Los valores posibles son <code>ScrollBarPolicy.ALWAYS</code> , <code>ScrollBarPolicy.AS_NEEDED</code> , <code>ScrollBarPolicy.NEVER</code> .

## ColorPicker

Este control posibilita la selección de un color. Su comportamiento consiste en mostrar una paleta de colores predefinidos al usuario, una vez que el usuario hace su elección la paleta se oculta y se puede obtener el color seleccionado. También puede mostrar una ventana emergente para que el usuario cree un color a sus necesidades.

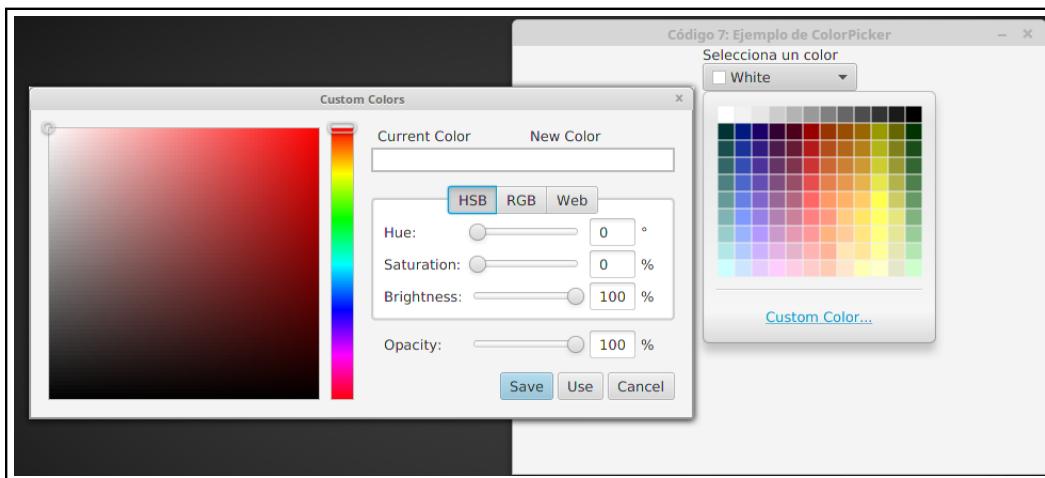


Figura 1.13: Ventana emergente para selección de colores

## ScrollPane

Al igual que TabPane, ScrollPane es un control y no un panel, y ofrece un área desplazable de visualización de su contenido.

## SplitPane

Este control puede contener dos o más secciones acomodadas vertical u horizontalmente, separando las secciones con divisores que pueden ser arrastrados para dar más espacio a una de las secciones.

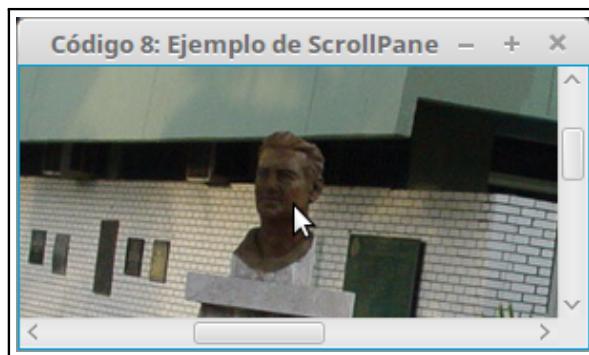


Figura 1.14: Ejemplo de una instancia de ScrollPane que contiene una imagen



Figura 1.15: Ejemplo del uso de SplitPane que contiene tres paneles distintos

Tabla 1.9: Métodos útiles para la personalización de la apariencia de controles

Método	Clase	Descripción
setStyle(String s)	javafx.scene.Node	Modifica la propiedad visual del control, su sintaxis es 'propiedad : valor' .
getStylesheets()	javafx.scene.Parent	Devuelve la lista URLs de <b>archivos</b> CSS asociados al nodo, a esa lista se pueden añadir o quitar URLs.
getStyleClass()	javafx.css.Styleable	Devuelve la lista de clases asociadas al control, a esa lista se pueden añadir o quitar clases CSS.

## ToolBar

Las barras de herramientas facilitan al usuario el acceso a aquellas funciones que se utilizan con mayor frecuencia en las aplicaciones. Para tal propósito JavaFX tiene el control ToolBar al que se pueden añadir instancias de cualquier subclase de Node.

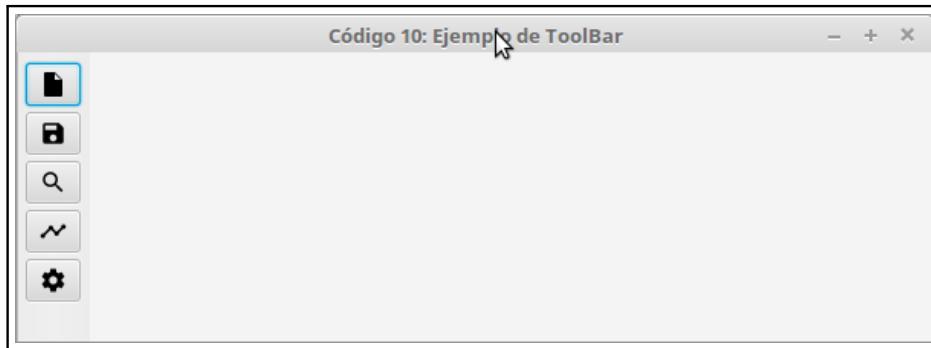


Figura 1.16: Ejemplo del uso de ToolBar con orientación vertical

## Apariencia de la Aplicación

Con JavaFX la presentación visual de los controles de la interface es muy personalizable gracias a las reglas CSS. Se tienen dos alternativas para modificar la apariencia de los controles, la primera es mediante una modificación programática de las propiedades visuales de los controles a través del métodos heredados por la clase `javafx.scene.Node` y la interfaz `javafx.css.Styleable`; la segunda alternativa la asociación de clases CSS mediante el atributo `class` de los elementos ligados a los controles en la definición de interfaces mediante archivos FXML. Detalles de la segunda alternativa se explican en la sección *Scene Builder*.

## Scene Builder

Una de las mejoras más sobresalientes incluidas en JavaFX es el diseño de interfaces gráficas mediante el lenguaje de marcado FXML, manera similar en la que se construyen interfaces

en los estándares web mediante HTML. FXML es un lenguaje derivado de XML y significa Fx eXtensible Markup Language, fue incluido en el conjunto de tecnologías que acompañan a JavaFX desde su versión 2.0.

Para facilitar la actividad de construcción de interfaces Oracle lanzó Scene Builder, una herramienta de diseño mediante un editor gráfico. Actualmente Oracle sólo proporciona Scene Builder como código fuente mediante el proyecto OpenJFX. Sin embargo, la empresa Gluon dedicada a la creación de soluciones y herramientas basadas en Java, mantiene soporte a una versión de Scene Builder. Scene Builder de Gluon<sup>1</sup> puede ser utilizado como una aplicación independiente o integrarse como plugin a los entornos de desarrollo NetBeans, Eclipse e IntelliJ.

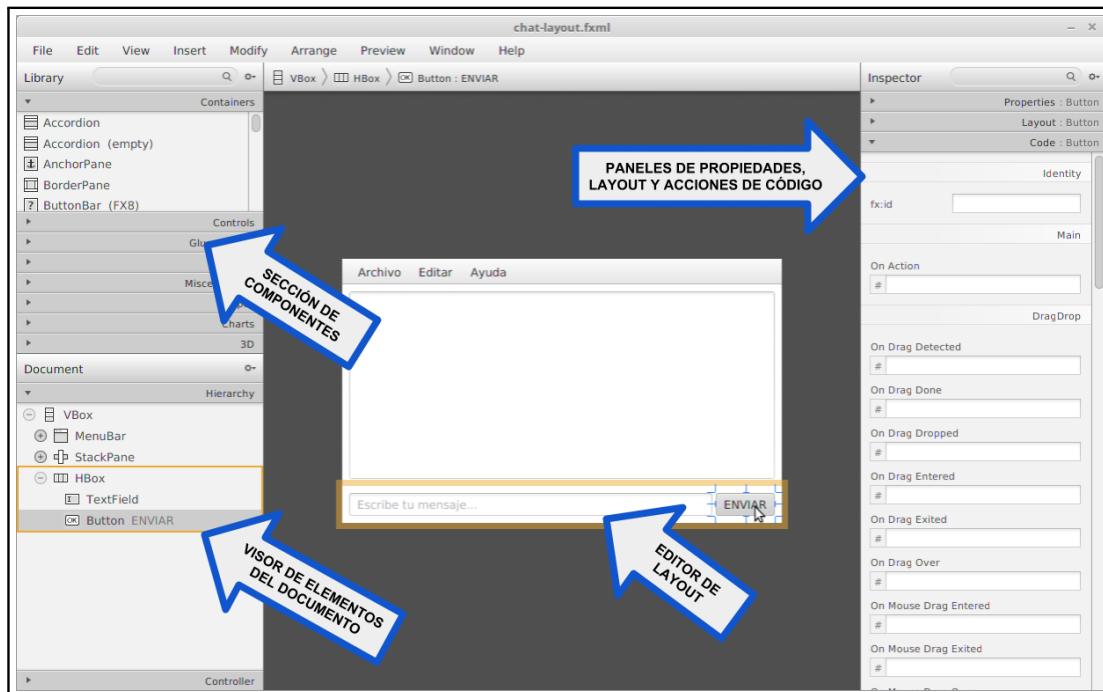


Figura 1.17: Interfaz de Scene Builder 2.0

<sup>1</sup><http://gluonhq.com/products/scene-builder/>

## Ejemplo de una aplicación de chat

La figura 1.18 muestra una interface para un chat creado con Scene Builder.

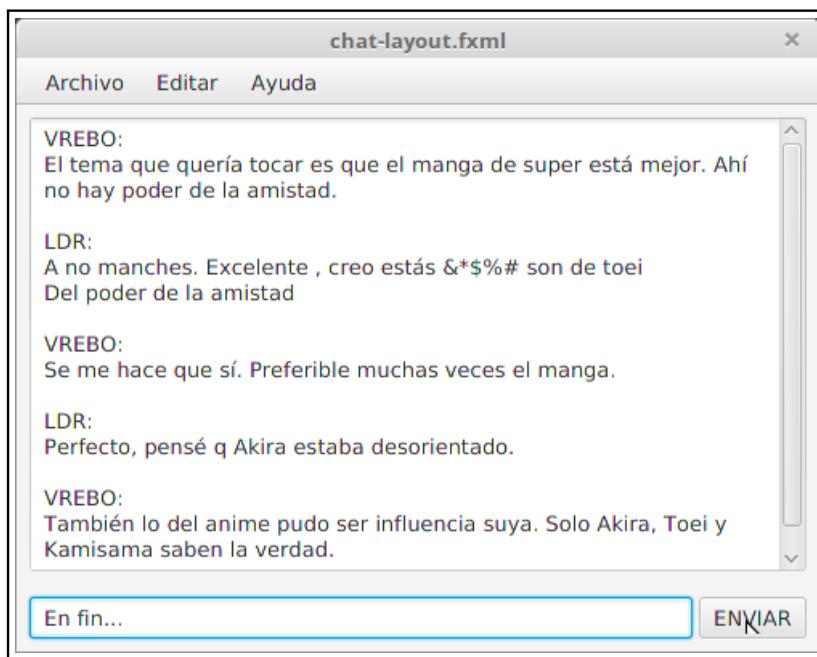


Figura 1.18: Captura de pantalla a la interfaz del chat

Código 1.9: Clase principal donde se carga el archivo FXML.

```
public class Main extends Application {
    public void start(Stage primaryStage) {
        Parent root;
        try {
            root = FXMLLoader.load(getClass().getResource("chat-layout.fxml"));
        } catch (IOException | NullPointerException e) {
            e.printStackTrace();
            root = new StackPane(new Label("Lo sentimos, hubo un error cargando el layout"));
        }
        primaryStage.setTitle("Demostración de uso de FXML");
        primaryStage.setScene(new Scene(root, 600, 300));
        primaryStage.show();
    }
    public static void main(String [] args) { launch(args); }
}
```

Código 1.10: Archivo FXML creado Scene Builder para la interface del chat.

```
<?xml version="1.0" encoding="UTF-8"?>
<?import javafx.geometry.Insets?>
<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Menu?>
<?import javafx.scene.control.MenuBar?>
<?import javafx.scene.control.MenuItem?>
<?import javafx.scene.control.SeparatorMenuItem?>
<?import javafx.scene.control.TextArea?>
<?import javafx.scene.control.TextField?>
<?import javafx.scene.layout.HBox?>
<?import javafx.scene.layout.StackPane?>
<?import javafx.scene.layout.VBox?>

<VBox maxHeight="-Infinity" maxWidth="-Infinity"
       minHeight="-Infinity" minWidth="-Infinity"
       prefHeight="400.0" prefWidth="600.0"
       xmlns="http://javafx.com/javafx/8.0.111" xmlns:fx="http://javafx.com/fxml/1">
    <children>
        <MenuBar>
            <menus>
                <Menu text="_Archivo">
                    <items>
                        <MenuItem mnemonicParsing="false" text="Abrir" />
                        <SeparatorMenuItem mnemonicParsing="false" />
                        <MenuItem mnemonicParsing="false" text="Salir" />
                    </items>
                </Menu>
                <Menu text="_Editar">
                    <items>
                        <MenuItem mnemonicParsing="false" text="Delete" />
                    </items>
                </Menu>
                <Menu text="A_yuda">
                    <items>
                        <MenuItem mnemonicParsing="false" text="About" />
                    </items>
                </Menu>
            </menus>
        </MenuBar>
        <StackPane VBox.vgrow="ALWAYS">
            <padding>
                <Insets bottom="8.0" left="8.0" right="8.0" top="8.0" />
            </padding>
            <children>
                <TextArea prefHeight="200.0" prefWidth="200.0" />
            </children>
        </StackPane>
        <HBox spacing="4.0" VBox.vgrow="NEVER">
            <children>
                <TextField promptText="Escribe_tu_mensaje..." HBox.hgrow="ALWAYS" />
                <Button mnemonicParsing="false" text="ENVIAR" />
            </children>
            <padding>
                <Insets bottom="8.0" left="8.0" right="8.0" top="8.0" />
            </padding>
        </HBox>
    </children>
</VBox>
```

## **Anexo D**

# **Computación Gráfica 1**

Los temas incluidos en este documento son:

1. Esquema General de Elementos Gráficos
2. Canvas
3. Contexto Gráfico
4. Primitivas Gráficas
5. Texto e Imágenes

# Unidad 1

## Elementos de Interfaces Gráficas

### Computación Gráfica

#### Esquema general de elementos gráficos

Para ubicar los objetos gráficos se utiliza un sistema de coordenadas. En computación gráfica, el sistema de coordenadas es disinto al tradicional. Vease la imagen 1.2 para una comparativa entre ambos sistemas.

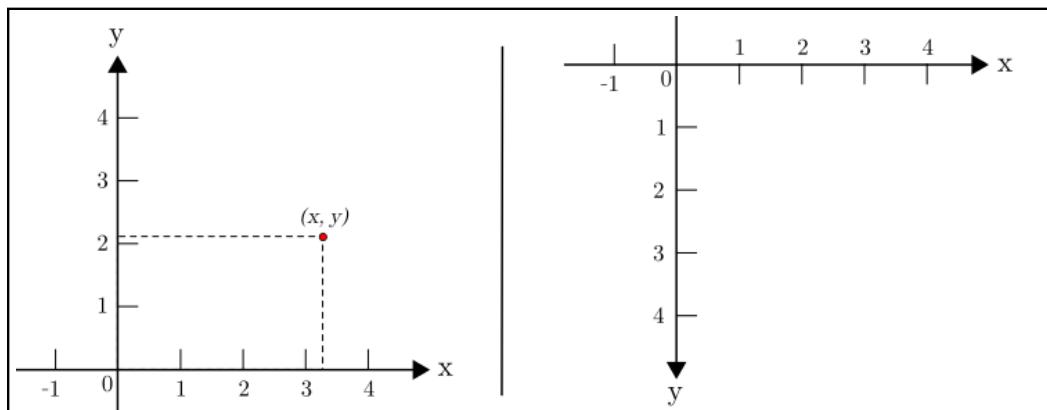


Figura 1.1: Izquierda: Sistema coordenado tradicional; Derecha: Sistema usado en computación gráfica

#### Prism

Prism es el componente de la arquitectura de JavaFX encargado de los trabajos de renderizado de gráficas. Posee un hilo de ejecución independiente para el renderizado, lo cual permite que un frame N sea renderizado mientras se está procesando el frame N+1 favoreciendo el desempeño general de las aplicaciones.

## Pulsos

*«A pulse is an event that indicates to the JavaFX scene graph that it is time to synchronize the state of the elements on the scene graph with Prism.»*

Java Documentation of Oracle<sup>1</sup>

En pocas palabras, cuando un pulso sucede, JavaFX se encarga de renderizar la sección o el elemento gráfico que disparó el pulso. Hay que resaltar que los programadores no tienen que lidiar directamente con los pulsos, estos eventos son generados tras bambalinas por el propio sistema de gráficos de JavaFX.

Un ejemplo de un evento desencadenador de un pulso es el cambio de ubicación de un control o la adición o remoción de un nodo (con una parte gráfica) a un gráfo de escena vivo<sup>2</sup>.

## Canvas

La clase `javafx.scene.canvas.Canvas` es una subclase de `Node` que funciona como un lienzo para trazar primitivas gráficas y dibujos en 2D, inclusive se pueden aplicar transformaciones a los trazos. La parte interna del canvas en que se pueden realizar los trazos se conoce como **contexto gráfico**.

- En JavaFX se utiliza un objeto como motor de interpretación de los trazos en el contexto gráfico (también se le conoce como brocha). La brocha es un objeto de la clase `javafx.scene.canvas.GraphicsContext`.
- El método `getGraphicsContext2D()` sirve para generar una brocha del canvas.

Código 1.1: Estructura general para el manejo del contexto gráfico.

```
import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;

public class MyCanvas extends Canvas{

    public MyCanvas(double width, double height) {
        super(width, height);
        draw();
    }

    public final void draw() {
        GraphicsContext gc = this.getGraphicsContext2D();
        ...
    }
}
```

<sup>1</sup><http://docs.oracle.com/javase/8/javafx/get-started-tutorial/jfx-architecture.htm#JFXST788>

<sup>2</sup>Un grafo de escena vivo es un grafo que está añadido a un objeto Stage.

## Contexto Gráfico

Como se mencionó previamente, el uso de un hilo de ejecución independiente da la ventaja de que, por un lado se puedan procesar los trazos mientras que por otro se realiza el renderizado de los gráficos.

La clase `GraphicsContext` realiza los trazos del canvas mediante un buffer<sup>3</sup>. Cada ocasión que se usa alguno de sus métodos de trazado, se colocan en el buffer los parámetros necesarios para renderizarse en la imagen del canvas al final de cada **pulso** en el hilo de renderizado.

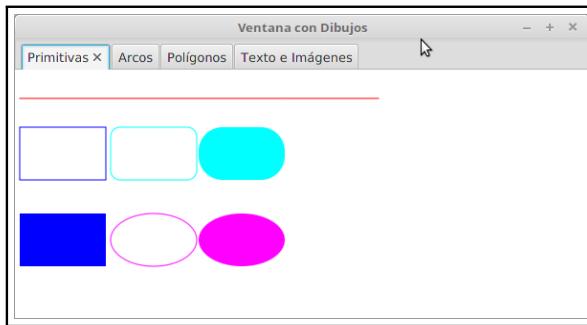


Figura 1.2: Ejemplo de trazado de primitivas gráficas.

Código 1.2: Aplicación de demostración para el manejo del contexto gráfico.

```
public class Main extends Application {

    public static final double WIDTH = 600;
    public static final double HEIGHT = 600;

    @Override
    public void start(Stage primaryStage) {
        Parent root = new GraphicContextPane(WIDTH, HEIGHT);
        primaryStage.setTitle("Ventana con Dibujos");
        primaryStage.setScene(new Scene(root, WIDTH, HEIGHT));
        primaryStage.show();
    }

    public static void main(String[] args) { launch(args); }
}
```

<sup>3</sup>Un buffer de datos es un espacio de la memoria en un disco o en un instrumento digital reservado para el almacenamiento temporal de información digital, mientras que está esperando ser procesada. - Wikipedia

Código 1.3: Clase auxiliar para la construcción de la interface.

```
public class GraphicContextPane extends TabPane {

    public GraphicContextPane(double width, double height) {
        setBackground(new Background(new BackgroundFill(Color.WHITE, null, null)));
        getTabs().add(new Tab("Primitivas", new PrimitivesPane(width, height)));
        getTabs().add(new Tab("Arcos", new ArcsPane(width, height)));
        getTabs().add(new Tab("Polígonos", new PolygonsPane(width, height)));
        getTabs().add(new Tab("Texto_e_Imágenes", new TextImagesPane(width, height)));
    }
}
```

## Primitivas Gráficas

La clase GraphicsContext proporciona los siguientes métodos para presentar las primitivas gráficas:

### Trazos

- void strokeLine(double x1, double y1, double x2, double y2)
- void strokeRect(double x, double y, double w, double h)
- void strokeOval(double x, double y, double w, double h)
- void strokeRoundRect(double x, double y, double w, double h, double arcWidth, double arcHeight)

### Figuras con relleno

- void fillRect(double x, double y, double w, double h)
- void fillOval(double x, double y, double w, double h)
- void fillRoundRect(double x, double y, double w, double h, double arcWidth, double arcHeight)

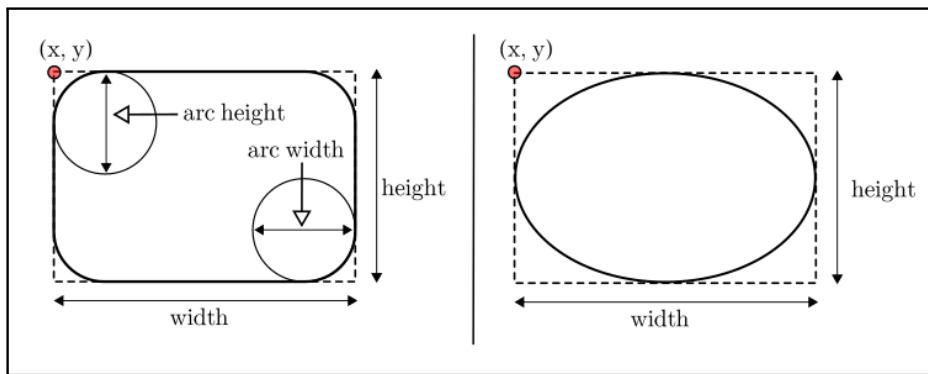


Figura 1.3: Izquierda: Cuadrado redondeado y sus parámetros; Derecha: Óvalo y sus parámetros.

## Color

Para definir un nuevo color de contorno o relleno de un trazo, se utilizan los métodos

**setStroke(Paint p)** y **setFill(Paint p)**

respectivamente.

- Se pueden utilizar los colores definidos en la clase `javafx.scene.paint.Color` (`RED`, `MAGENTA`, `CYAN`, `BLACK`, ..., entre muchos otros).
  - `gc.setStroke(Color.CYAN);`
  - `gc.setFill(Color.ORANGE);`
- Se puede crear un color a partir de sus componentes RGB (rojo, verde y azul) y su opacidad.
  - `gc.setStroke(new Color(0.4, 0.4, 0, 1));`
  - `gc.setFill(new Color(0.4, 0.4, 0, 1));`
- Los componentes de color y la opacidad tienen valores entre 0 y 1.

Código 1.4: Clase PrimitivesPane.

```
import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.paint.Color;

public class PrimitivesPane extends Canvas {

    public PrimitivesPane(double width, double height) {
        super(width, height);
        draw();
    }

    public final void draw() {
        GraphicsContext gc = getGraphicsContext2D();
        gc.setStroke(Color.RED);
        gc.strokeLine(5, 30, 380, 30);

        gc.setStroke(Color.BLUE);
        gc.strokeRect(5, 60, 90, 55);
        gc.setFill(Color.BLUE);
        gc.fillRect(5, 150, 90, 55);

        gc.setStroke(Color.CYAN);
        gc.strokeRoundRect(100, 60, 90, 55, 20, 20);
        gc.setFill(Color.CYAN);
        gc.fillRoundRect(192, 60, 90, 55, 50, 50);

        gc.setStroke(Color.MAGENTA);
        gc.strokeOval(100, 150, 90, 55);
        gc.setFill(Color.MAGENTA);
        gc.fillOval(192, 150, 90, 55);
    }
}
```

## Arcos

- void strokeArc(double x, double y, double w, double h, double sa, double aa, ArcType closure)
- void fillArc(double x, double y, double w, double h, double sa, double aa, ArcType closure)

**ArcType** Indica como será el cierre del arco.

ROUND: Cierra el arco con líneas que parten desde los puntos de inicio y fin y se conectan en el centro del arco.

CORD: Cierra el arco con una línea recta que conecta los puntos de inicio y fin.

OPEN: No cierra el arco.

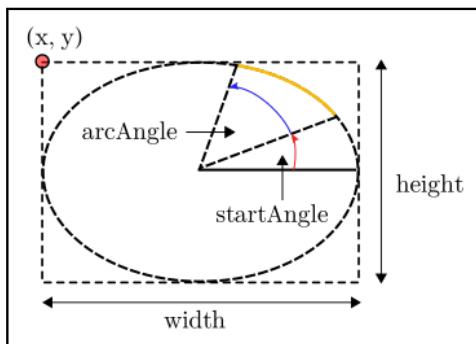


Figura 1.4: Arco y sus parámetros.

Código 1.5: Clase ArcsPane.

```
...
import javafx.scene.shape.ArcType;

public class ArcsPane extends Canvas {

    public ArcsPane(double width, double height) {
        super(width, height); draw();
    }

    public final void draw() {
        GraphicsContext gc = getGraphicsContext2D();

        gc.setStroke(Color.RED);
        gc.strokeRect(40, 35, 80, 80);
        gc.strokeRect(160, 35, 80, 80);
        gc.strokeRect(280, 35, 80, 80);
        gc.setStroke(Color.BLACK);
        gc.strokeArc(40, 35, 80, 80, 0, 360, ArcType.ROUND);
        gc.strokeArc(160, 35, 80, 80, 0, 110, ArcType.ROUND);
        gc.strokeArc(280, 35, 80, 80, 0, -270, ArcType.ROUND);
        gc.setFill(Color.ORANGE);
        gc.fillArc(40, 150, 80, 70, 0, 360, ArcType.ROUND);
        gc.fillArc(160, 150, 80, 70, 270, -90, ArcType.ROUND);
        gc.fillArc(280, 150, 80, 70, 0, -270, ArcType.ROUND);
    }
}
```

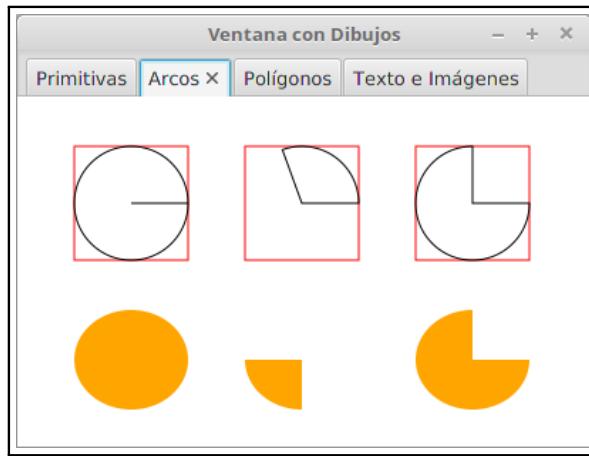


Figura 1.5: Canvas con arcos dibujados.

## Polígonos y Poli-líneas

- void strokePolygon(double[] xPoints, double[] yPoints, int nPoints)
- void fillPolygon(double[] xPoints, double[] yPoints, int nPoints)
- void strokeLine(double x1, double y1, double x2, double y2)

Código 1.6: Clase PolygonsPane.

```

...
public class PolygonsPane extends Canvas {

    public PolygonsPane(double width, double height) {
        super(width, height);
        draw();
    }

    public final void draw() {
        GraphicsContext gc = getGraphicsContext2D();
        double[] x1 = {50, 140, 160, 120, 100, 45};
        double[] y1 = {50, 100, 120, 190, 190, 60};
        gc.strokePolygon(x1, y1, x1.length);

        int incX = 200;
        int incY = 0;
        double[] x3 = {incX + 50, incX + 140, incX + 160, incX + 120, incX + 100, incX + 45};
        double[] y3 = {incY + 50, incY + 100, incY + 120, incY + 190, incY + 190, incY + 60};
        gc.setFill(Color.BLUE);
        gc.fillPolygon(x3, y3, 6);

        incY = 0;
        incX = 400;
        double[] x2 = {incX + 50, incX + 140, incX + 160, incX + 120, incX + 100, incX + 45};
        double[] y2 = {incY + 50, incY + 100, incY + 120, incY + 190, incY + 190, incY + 60};
        gc.setFill(Color.RED);
        gc.strokePolyline(x2, y2, 6);
    }
}

```

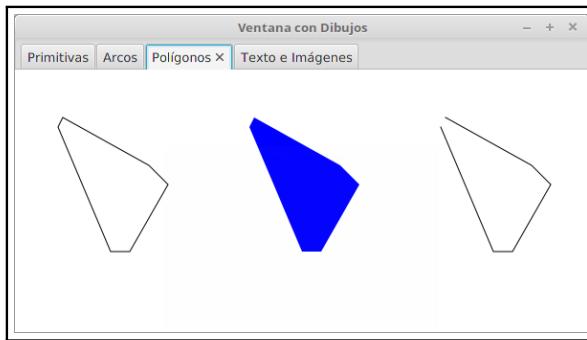


Figura 1.6: Canvas con polígonos y poli-línea.

## Texto e Imágenes

`GraphicsContext` es capaz de dibujar texto e imágenes.

- `void strokeText(String text, double x, double y)`
- `void strokeText(String text, double x, double y, double maxWidth)`
- `void fillText(String text, double x, double y)`
- `void fillText(String text, double x, double y, double maxWidth)`
- `void drawImage(Image img, double x, double y)`
- `void drawImage(Image img, double x, double y, double w, double h)`
- `void drawImage(Image img, double sx, double sy, double sw, double sh, double dx, double dy, double dw, double dh)`

Se puede especificar la fuente que se usará para escribir en el canvas mediante el método `setFont(Font f)`.

- Se puede crear una fuente a través de alguno de los constructores de la clase `Font`:
  - `Font(double size)`
  - `Font(String name, double size)`
- Se puede crear una fuente a través de alguno de los métodos estáticos de la clase `Font`:
  - `font(String family)`
  - `font(double size)`
  - `font(String family, double size)`
  - `font(String family, FontPosture p, double size)`
  - `font(String family, FontWeight w, double size)`
  - `font(String family, FontWeight w, FontPosture p, double size)`
- Se pueden usar las fuentes cargadas en la máquina o alguno de los cinco tipos de fuente base (`SansSerif`, `Serif`, `Monospaced`, `Dialog` y `DialogInput`).

## Tópicos Avanzados de Programación

## Unidad: Elementos de Interfaces Gráficas

- `FontWeight` es un enumerado que puede ser: `THIN`, `EXTRA_LIGHT`, `LIGHT`, `NORMAL`, `MEDIUM`, `SEMI_BOLD`, `BOLD`, `EXTRA_BOLD`, `BLACK`.
- `FontPosture` es un enumerado que puede ser: `ITALIC`, `NORMAL`.

Código 1.7: Clase `TextImagesPane`.

```

import javafx.scene.image.Image;
import javafx.scene.paint.Color;
import javafx.scene.text.Font;
import javafx.scene.text.FontPosture;
import javafx.scene.text.FontWeight;

public class TextImagesPane extends Canvas {

    public TextImagesPane(double width, double height) {
        super(width, height);
        draw();
    }

    public final void draw() {
        GraphicsContext gc = getGraphicsContext2D();

        gc.setStroke(Color.BLUE);
        gc.strokeText("Mensaje_en_azul", 20, 20);

        gc.setStroke(new Color(0.4, 0.4, 0, 1));
        gc.strokeText("Mensaje_en_otro_color", 400, 20);

        Font font;
        font = Font.font("Arial", FontWeight.BOLD, 24);
        gc.setFont(font);
        gc.setStroke(Color.RED);
        gc.strokeText("Mensaje_en_Arial", 20, 350);

        font = Font.font("Courier", FontWeight.BOLD, FontPosture.ITALIC, 18);
        gc.setStroke(Color.MAGENTA);
        gc.setFont(font);
        gc.strokeText("Mensaje_en_Courier", 400, 350);

        Image img = new Image(getClass().getResourceAsStream("/images/duke.jpg"));
        double imgWidth = img.getWidth();
        double imgHeight = img.getHeight();

        double canvasWidth = this.getWidth();
        double canvasHeight = this.getHeight();

        double x = canvasWidth / 2 - imgWidth / 2;
        double y = canvasHeight / 2 - imgHeight / 2;

        gc.drawImage(img, x, y);
    }
}

```

Tópicos Avanzados de Programación

Unidad: Elementos de Interfaces Gráficas



Figura 1.7: Canvas con texto e imagen.

## **Anexo E**

### **Computación Gráfica 2**

Se presentan los ejemplos con primitivas gráficas.

1. Ejemplo para graficar un símbolo de peligro radioactivo.
2. Ejemplo de un dibujo compuesto con varias primitivas.

# Unidad 1

## Elementos de Interfaces Gráficas

### Computación Gráfica

#### Ejemplos de uso de primitivas de dibujo

##### Proyecto Símbolo

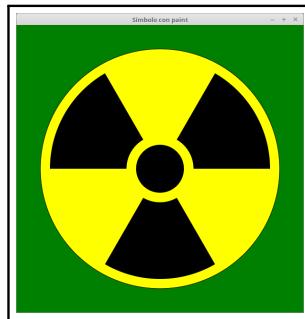


Figura 1.1: Señal de riesgo por radiación ionizante dibujado en el contexto gráfico.

Código 1.1: Aplicación para dibujar el símbolo de riesgo radiactivo.

```
public class Main extends Application {  
  
    private static final int WIDTH = 600;  
    private static final int HEIGHT = 600;  
  
    @Override  
    public void start(Stage primaryStage) {  
        Parent root = new SymbolPane(WIDTH, HEIGHT);  
        primaryStage.setTitle("Símbolo con paint");  
        primaryStage.setScene(new Scene(root, WIDTH, HEIGHT));  
        primaryStage.show();  
    }  
  
    public static void main(String[] args) { launch(args); }  
}
```

**Método translate**

Mueve el centro del sistema de coordenadas al punto especificado.

- void translate(double x, double y)

Código 1.2: Clase RadioactiveSymbol.

```
public class RadioactiveSymbol extends Pane {

    private final Canvas canvas;

    public RadioactiveSymbol(double width, double height) {
        canvas = new Canvas(width, height);
        init();
        draw();
    }

    private void init() {
        getChildren().add(canvas);
        setBackground(new Background(
            new BackgroundFill(
                Color.GREEN, CornerRadii.EMPTY, Insets.EMPTY)));
    }

    public final void draw() {
        GraphicsContext gc = canvas.getGraphicsContext2D();

        // Obtener las dimensiones del panel
        double width = canvas.getWidth();
        double height = canvas.getHeight();

        // Mover el origen al centro del panel
        gc.translate(width / 2, height / 2);

        int exteriorR = 250; //Radio del circulo externo (circulo amarillo)
        gc.setFill(Color.YELLOW);
        gc.fillOval(-exteriorR, -exteriorR, 2 * exteriorR, 2 * exteriorR);
        gc.setStroke(Color.BLACK);
        gc.strokeOval(-exteriorR, -exteriorR, 2 * exteriorR, 2 * exteriorR);

        int arcR = 230; //Radio de los arcos para las hélices.
        gc.setFill(Color.BLACK);
        gc.fillArc(-arcR, -arcR, arcR * 2, arcR * 2, 0, 60, ArcType.ROUND);
        gc.fillArc(-arcR, -arcR, arcR * 2, arcR * 2, 120, 60, ArcType.ROUND);
        gc.fillArc(-arcR, -arcR, arcR * 2, arcR * 2, 240, 60, ArcType.ROUND);

        int interiorR = 70; //Radio del circulo interior (circulo amarillo)
        gc.setFill(Color.YELLOW);
        gc.fillOval(-interiorR, -interiorR, interiorR * 2, interiorR * 2);

        int centerR = 50; //Radio del circulo interior (circulo negro)
        gc.setFill(Color.BLACK);
        gc.fillOval(-centerR, -centerR, centerR * 2, centerR * 2);
    }
}
```

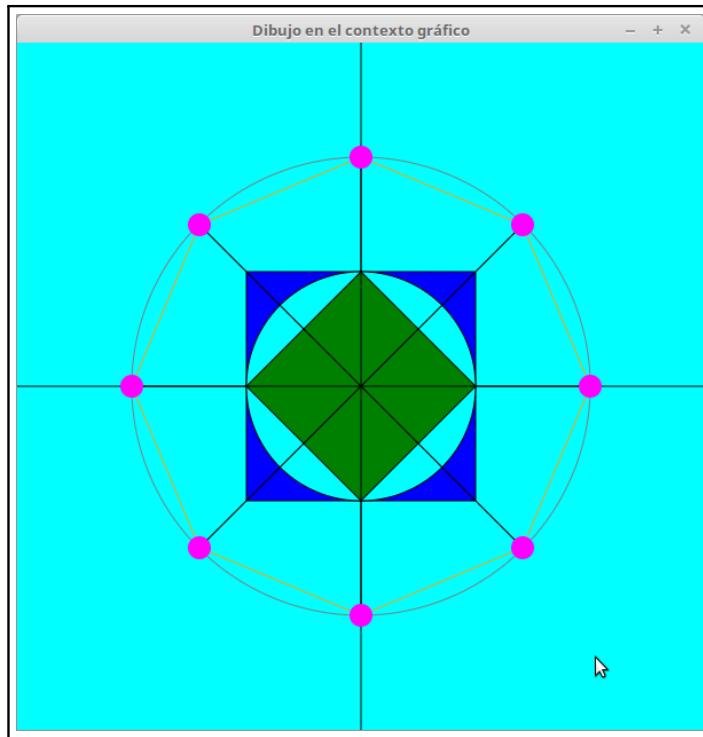
**Proyecto Dibujo**

Figura 1.2: Dibujo de figuras en el contexto gráfico.

Código 1.3: Aplicación para dibujar figuras en el contexto gráfico.

```
public class Main extends Application {  
  
    private static final int WIDHT = 600;  
    private static final int HEIGHT = 600;  
  
    @Override  
    public void start(Stage primaryStage) {  
        Parent root = new DrawPane(WIDHT, HEIGHT);  
        primaryStage.setTitle("Dibujo en el contexto gráfico");  
        primaryStage.setScene(new Scene(root, WIDHT, HEIGHT));  
        primaryStage.show();  
    }  
  
    public static void main(String[] args) { launch(args); }  
}
```

Código 1.4: Clase DrawPane (Parte 1).

```

public class DrawPane extends Pane {

    private final Canvas canvas;

    public DrawPane(double width, double height) {
        canvas = new Canvas(width, height);
        init();
        draw();
    }

    private void init() {
        getChildren().add(canvas);
        setBackground(new Background(
            new BackgroundFill(
                Color.CYAN, CornerRadii.EMPTY, Insets.EMPTY)));
    }

    private void draw() {
        GraphicsContext gc = canvas.getGraphicsContext2D();

        // Obtener las dimensiones del panel
        double width = canvas.getWidth();
        double height = canvas.getHeight();

        // Mover el origen al centro
        gc.translate(width / 2, height / 2);

        // Dibujar los ejes de coordenadas
        gc.strokeLine(-width / 2, 0, width / 2, 0);
        gc.strokeLine(0, -height / 2, 0, height / 2);

        // Cuadrado con centro en el origen
        int sidelenght = 200;
        gc.setFill(Color.BLUE);
        gc.fillRect(-sidelenght / 2, -sidelenght / 2, sidelenght, sidelenght);
        gc.setStroke(Color.BLACK);
        gc.strokeRect(-sidelenght / 2, -sidelenght / 2, sidelenght, sidelenght);

        // Círculo con centro en el origen
        gc.setFill(Color.CYAN);
        gc.fillOval(-sidelenght / 2, -sidelenght / 2, sidelenght, sidelenght);
        gc.strokeOval(-sidelenght / 2, -sidelenght / 2, sidelenght, sidelenght);

        // Rombo inscrito en el cuadrado
        double[] px = {0, -sidelenght / 2, 0, sidelenght / 2};
        double[] py = {-sidelenght / 2, 0, sidelenght / 2, 0};
        gc.setFill(Color.GREEN);
        gc.fillPolygon(px, py, px.length);
        gc.strokePolygon(px, py, px.length);

        // Círculo con centro en el origen
        int radious = 200;
        gc.setStroke(Color.GRAY);
        gc.strokeOval(-radious, -radious, 2 * radious, 2 * radious);

        // Dibujar rayos del centro a ocho puntos en el círculo
        gc.setStroke(Color.BLACK);
        for (int i = 0; i < 8; i++) {
            double rad = Math.toRadians(45 * i);
            int x = (int) (radious * Math.cos(rad));
            int y = (int) (radious * Math.sin(rad));
            gc.strokeLine(0, 0, x, y);
        }
        ...
    }
}

```

Código 1.5: Clase DrawPane (Parte 2).

```
...
// Dibujar líneas que unan los rayos
gc.setStroke(Color.ORANGE);
int ax = radious;
int ay = 0;
for (int i = 0; i <= 8; i++) {
    double rad = Math.toRadians(45 * i);
    int x = (int) (radious * Math.cos(rad));
    int y = (int) (radious * Math.sin(rad));
    gc.strokeLine(ax, ay, x, y);
    ax = x;
    ay = y;
}

// Dibujar 8 puntos al final de cada rayo
gc.setFill(Color.MAGENTA);
for (int i = 0; i < 8; i++) {
    double rad = Math.toRadians(45 * i);
    int x = (int) (radious * Math.cos(rad));
    int y = (int) (radious * Math.sin(rad));
    gc.fillOval(x - 10, y - 10, 20, 20);
}
}
```

## **Anexo F**

### **Computación Gráfica 3**

Se presentan un ejemplo del uso de flujos de archivos para lectura de puntos en el contexto gráfico y su posterior graficación formando un polígono.

# Unidad 1

## Elementos de Interfaces Gráficas

### Computación Gráfica

#### Conceptos sobre archivos

Desde una perspectiva de "bajo nivel" se puede definir un archivo como: Un conjunto de bits almacenados en un dispositivo y accesible a través de una ruta de acceso (path) que lo identifica. En general, existen dos criterios para clasificar a los archivos.

- **De acuerdo a su contenido:** Los archivos de caracteres (o de texto) y los de bytes (binarios).
  - **Archivo de texto:** Es aquél formado exclusivamente por caracteres y pueden crearse y visualizarse usando un editor (los archivos de código fuente de Java).
  - **Archivo binario:** No está formado por caracteres si no por los bytes que contiene y pueden representar imágenes, sonido, etc.
- **De acuerdo de acceso:** Acceso secuencial o acceso directo.
  - **Acceso secuencial:** La información del archivo es una secuencia de elementos (bytes o caracteres) de manera que para acceder al  $i$ -ésimo elemento se debe haber accedido a los  $i-1$  elementos anteriores.
  - **Acceso directo:** La información del archivo puede ser accesada de forma directa a través de apuntadores o índices.

#### Ler archivos desde Java

Se utilizan las clase del paquete `java.io` para manipulación de archivos. El código que maneja archivos debe considerar que varias cosas pueden fallar al tratar de manipularlos, por ejemplo: el archivo está dañado, desconexión inesperada de la fuente de datos, etc.

### Manejo de excepciones

Las excepciones son un mecanismo que permite a los métodos indicar si algún error ha sucedido (una situación excepcional), de manera que quien ha invocado al método puede detectar la situación errónea y actuar acorde al caso.

Cuando un error sucede, el método lanza (throw) una excepción y en lugar de seguir la ejecución normal de instrucciones, se busca hacia atrás en la secuencia de llamadas si existe alguna que pueda atraparla (catch). Si no se pueden atrapar, el programa acaba su ejecución y se informa del error que ha producido la excepción.

Las excepciones pueden ser:

- **Excepciones de tiempo de ejecución:** Estas no obligan al programador a tratarlas explícitamente.
- **Excepciones verificadas:** Obligan al programador a atrapar la excepción (bloque `try-catch`) o indicar que dicho método puede lanzar la excepción (declaración `throws`).

### Lectura de archivos secuenciales de texto

- La lectura de un archivo se realiza utilizando **flujos** o streams.
- **FileReader** permite leer caracteres de un archivo.
- El método `tread()` lee el siguiente carácter no leído del archivo.
- El método `read()` devuelve -1 cuando ya no hay más caracteres que leer.

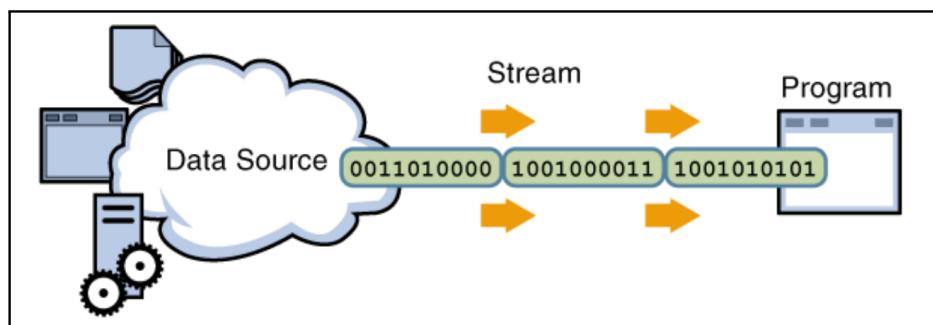


Figura 1.1: Esquema general del flujo de archivos.

### Buffering

Para leer elementos en bloque se requiere de un buffer (o memoria temporal) que permita almacenar los caracteres hasta que se cumpla una condición, un salto de línea por ejemplo.

## Tópicos Avanzados de Programación

## Unidad: Elementos de Interfaces Gráficas

- **BufferedReader** es un buffer para leer líneas de caracteres y almacenarlos en objetos String.
- BufferedReader no puede leer directamente de un archivo, si no que requiere de un objeto.
- El método **readLine()** permite leer una cadena de caracteres completa hasta encontrar la marca de fin de línea.
- El método **readLine ()** no regresa null cuando no puede leer más líneas.

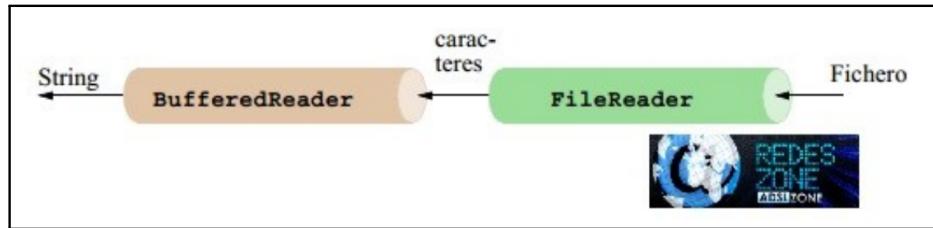


Figura 1.2: Esquema detallado del flujo de archivos.

Los elementos leídos por FileReader y/o BufferedReader deben almacenarse en algún objeto.

- **Estructura estática (String):** Antes de leer el archivo se requiere conocer el número de elementos a leer (caracteres o líneas). Si no se conocen, primero se deben contar los elementos y luego almacenarlos.

```
String[] lines = new String[ELEMENTS_COUNT];
```

- **Estructura dinámica (Listas enlazadas):** Al leer un elemento, se va adicionando a la estructura, qué va creciendo conforme se adicionan elementos.

```
List<String> lines = new ArrayList<>();
```

Código 1.1: Clase auxiliar para funciones con archivos.

```
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

public class FileHelper {

    public static List<String> readFile(String fileName) {

        List<String> lines = new ArrayList<>();

        try { //Flujo de caracteres
            FileReader stream = new FileReader(fileName);
            //Buffer de Strings
            BufferedReader reader = new BufferedReader(stream)) {

                String line = reader.readLine();
                while (line != null) { //Si line == null se alcanzó el final del archivo
                    lines.add(line);
                    line = reader.readLine();
                }
            } catch (FileNotFoundException ex) {
                System.err.println("Archivo_no_encontrado");
                System.err.println("Detalles:_ " + ex.getMessage());
            } catch (IOException ex) {
                System.err.println("Error_con_el_flujo_del_archivo");
                System.err.println("Detalles:_ " + ex.getMessage());
            }
        }

        return lines;
    }
}
```

## Aplicación para desplegar el contenido de un archivo

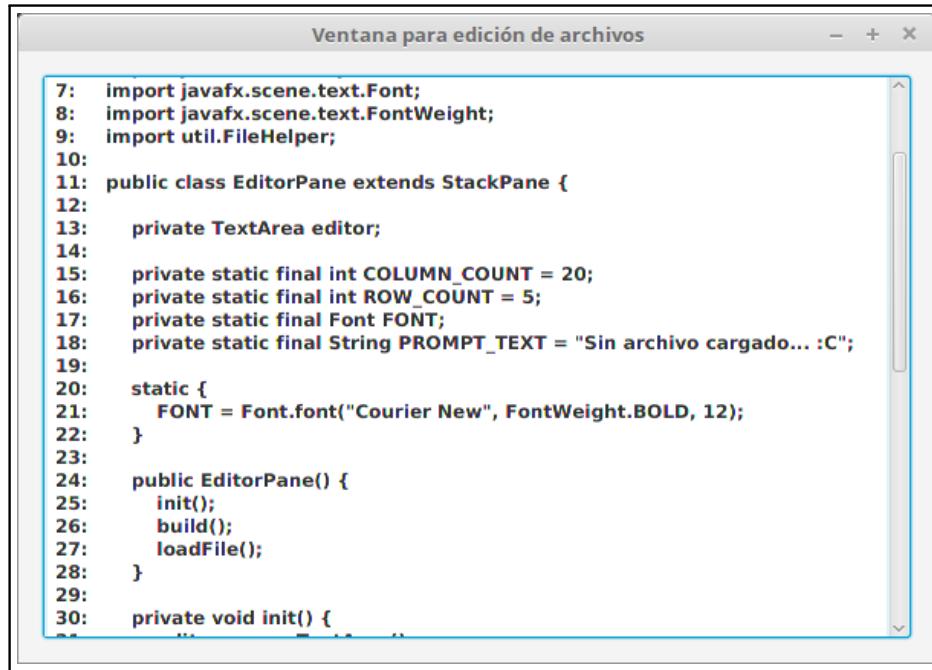


Figura 1.3: Área de texto que muestra el código leído desde un archivo

Código 1.2: Aplicación del editor que lee un archivo.

```
public class Main extends Application {
    @Override
    public void start(Stage primaryStage) {
        Parent root = new EditorPane();
        primaryStage.setTitle("Ventana para edición de archivos");
        primaryStage.setScene(new Scene(root, 600, 400));
        primaryStage.show();
    }

    public static void main(String[] args) { launch(args); }
}
```

## Código 1.3: Clase EditorPane.

```

import java.util.List;
import javafx.geometry.Insets;
import javafx.scene.control.TextArea;
import javafx.scene.layout.StackPane;
import javafx.scene.text.Font;
import javafx.scene.text.FontWeight;
import util.FileHelper;

public class EditorPane extends StackPane {

    private TextArea editor;

    private static final int COLUMN_COUNT = 20;
    private static final int ROW_COUNT = 5;
    private static final Font FONT;
    private static final String PROMPT_TEXT = "Sin archivo cargado...:C";

    static {
        FONT = Font.font("Courier New", FontWeight.BOLD, 12);
    }

    public EditorPane() {
        init();
        build();
        loadFile();
    }

    private void init() {
        editor = new TextArea();
        setPadding(new Insets(16));
    }

    private void build() {
        editor.setPrefColumnCount(COLUMN_COUNT);
        editor.setPrefRowCount(ROW_COUNT);
        editor.setFont(FONT);
        editor.setPromptText(PROMPT_TEXT);
        getChildren().add(editor);
    }

    private void loadFile() {
        String file = "./src/code5/EditorPane.java";

        List<String> content = FileHelper.readFile(file);

        int i = 1;

        for (String line : content) {
            editor.appendText(i + ": " + line + "\n");
            i++;
        }
    }
}

```

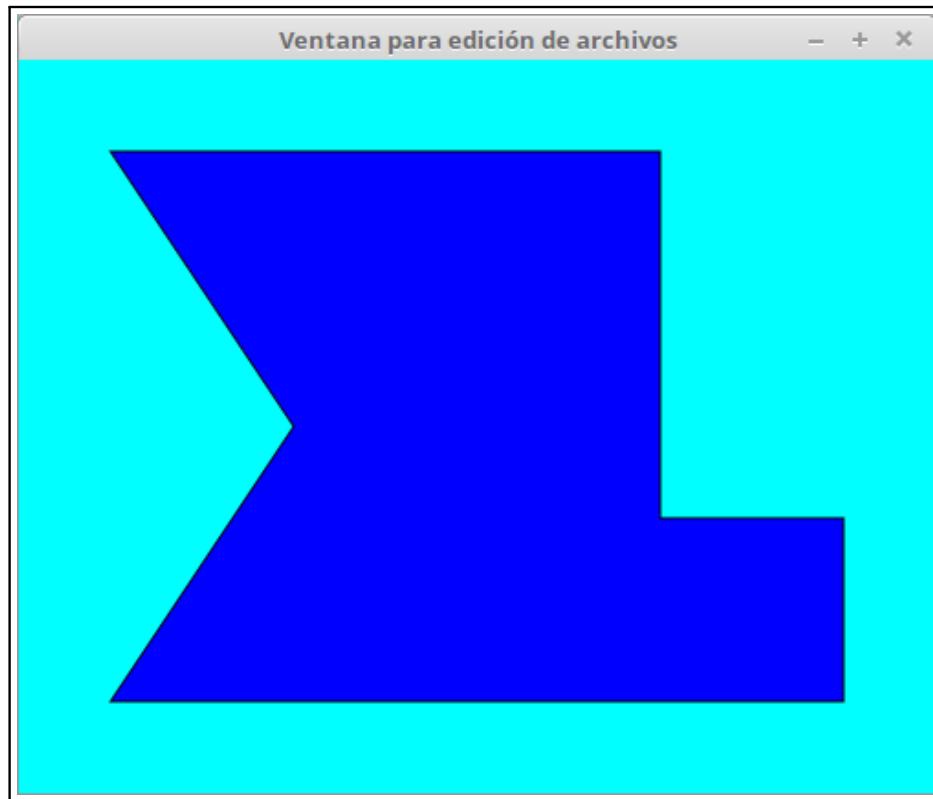
**Clases para leer y dibujar un polígono**

Figura 1.4: Polígono dibujado con coordenadas leídas desde un archivo

Código 1.4: Aplicación para dibujar figuras un polígono usando un archivo.

```
public class Main extends Application {  
    @Override  
    public void start(Stage primaryStage) {  
        Parent root = new PolygonPane();  
        primaryStage.setTitle("Dibujo de un polígono");  
        primaryStage.setScene(new Scene(root));  
        primaryStage.show();  
    }  
    public static void main(String[] args) { launch(args); }  
}
```

## Código 1.5: Clase PolygonPane.

```

import java.util.List;
import javafx.geometry.Insets;
import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.layout.Background;
import javafx.scene.layout.BackgroundImage;
import javafx.scene.layout.CornerRadii;
import javafx.scene.layout.StackPane;
import javafx.scene.paint.Color;
import util.FileHelper;

public class PolygonPane extends StackPane {

    private double[] x;
    private double[] y;

    private final Canvas canvas;

    public PolygonPane() {
        canvas = new Canvas(500, 400);
        init();
        loadFile();
        draw();
    }

    private void init() {
        setBackground(new Background(
            new BackgroundFill(Color.CYAN, CornerRadii.EMPTY, Insets.EMPTY)));
        getChildren().add(canvas);
    }

    private void draw() {
        if (x != null) {
            GraphicsContext gc = canvas.getGraphicsContext2D();
            gc.setFill(Color.BLUE);
            gc.fillPolygon(x, y, x.length);
            gc.setStroke(Color.BLACK);
            gc.strokePolygon(x, y, x.length);
        }
    }

    private void loadFile() {
        String file = "./src/code6/points.txt";

        List<String> lines = FileHelper.readFile(file);

        int pointsCount = lines.size();
        x = new double[pointsCount];
        y = new double[pointsCount];

        for (int i = 0; i < pointsCount; i++) {
            String linea = lines.get(i);
            String[] coord = linea.split(",");
            x[i] = Double.parseDouble(coord[0]);
            y[i] = Double.parseDouble(coord[1]);
        }
    }
}

```

## **Anexo G**

# **Programación Orientada a Eventos**

1. Concepto de Evento
2. Esquema de Gestión de Eventos
3. Clasificación de Eventos
4. Tipos de Oyentes de Eventos
5. Registro de Oyentes

# Unidad 1

## Elementos de Interfaces Gráficas

### Programación Orientada a Eventos

#### Concepto de Evento

- Un evento es resultado de la ejecución de una acción dentro de un entorno gráfico.
- Los eventos pueden ser generados por interacción del usuario con la GUI o por disparo mediante programación.
- Los eventos son objetos. En una ventana, los eventos representan acciones como hacer click a un botón, el movimiento del ratón, la captura de texto en un campo, etc.
- Cualquier interfaz gráfica constantemente monitorea los eventos en el sistema e informa de estos a los programas que se están ejecutando. Cada programa determina como reaccionar en respuesta a esos eventos.

#### Esquema del Manejo de Eventos

Para el manejo de eventos se requiere un esquema compuesto de los siguientes elementos **modelados como objetos**.

- **Origen del evento:** Típicamente son los componentes gráficos (botones, opciones de menú, cuadros de texto, etc) con los que interactúa el usuario.  
Al efectuar una acción sobre el componente (dar click, dar enter, etc) se **dispara** (crea) un objeto que encapsula dicha acción.
- **Evento:** Objeto que encapsula la información relacionada al objeto que lo originó (su nombre, por ejemplo) y las condiciones de la acción realizada.
- **Oyente del evento:** Cumple la función de "estar atentos escuchando" si se dispara un evento. Cuando un evento se dispara y es escuchado por el oyente, este puede ejecutar una o varias acciones para responder al evento.

Debido a que en una interfaz gráfica pueden existir múltiples orígenes de eventos, también pueden existir varios oyentes para atender de forma particular a cada origen. Esto tiene como beneficio que se delimitan las responsabilidades de cada oyente con mayor claridad.

Por lo anterior, es muy importante asociar a cada origen su correspondiente oyente. Esta operación se conoce como "registro del oyente al origen". Es claro que un oyente puede ser registrado a uno o varios componentes.

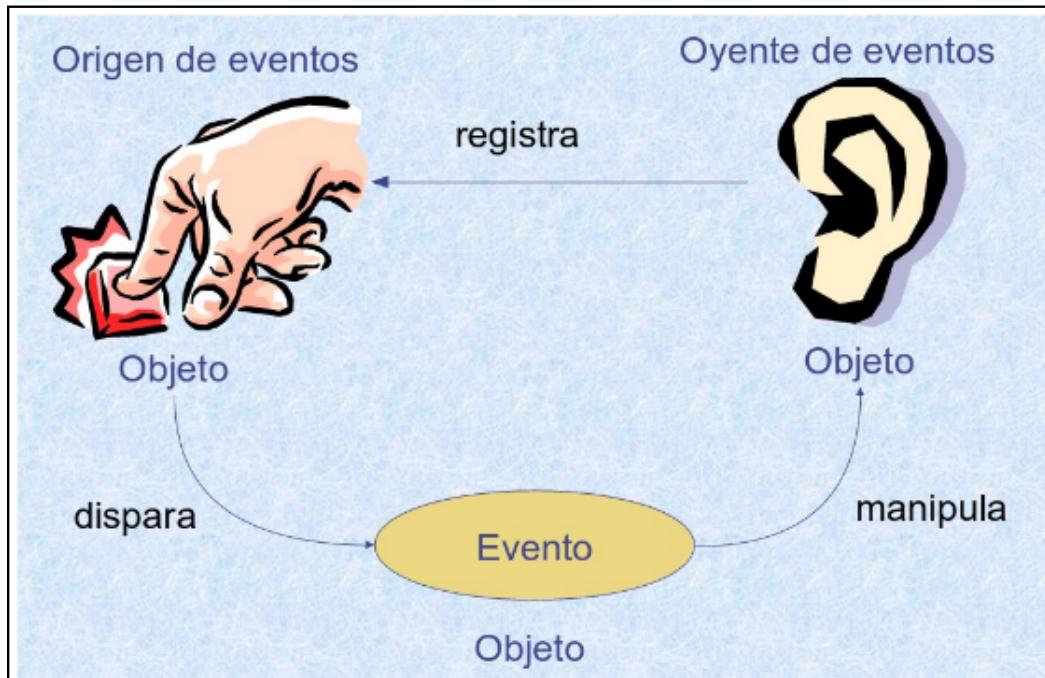


Figura 1.1: Esquema general para manejar los eventos

## Clasificación de Eventos

El esquema de eventos de JavaFX es distinto en diseño y funcionamiento al de sus predecesores Swing y AWT. Se integra de dos clases: `Event` e `EventType`, y una interface (clases con prototipos de métodos), `EventHandler`.

`Event` es la clase base que modela los eventos. Todos los eventos definidos en JavaFX extienden de `Event`, véase la figura 1.2. Toda instancia de `Event` tendrá asociado su origen (objeto que desencadenó el evento), un target (camino que seguirá el evento al ser disparado) y uno o más subtipos que son modelados por la clase `EventType`.

Cada subclase de `Event` es en sí un tipo de evento distinto, los tipos modelados por `EventType` son útiles para clasificar específicamente eventos que pertenecen a una misma subclase de

**Event.** Por ejemplo, la clase `MouseEvent` que modela los eventos de mouse que suceden sobre un componente puede especificarse en los eventos: botón presionado, botón liberado, click en mouse, movimiento del mouse, etc. Esta forma de subclasificar eventos se emplea agregando atributos estáticos a la subclase de `Event` en cuestión, véase la figura 1.3. La tabla 1.1 tiene una relación de los eventos y sus subtipos más relevantes.

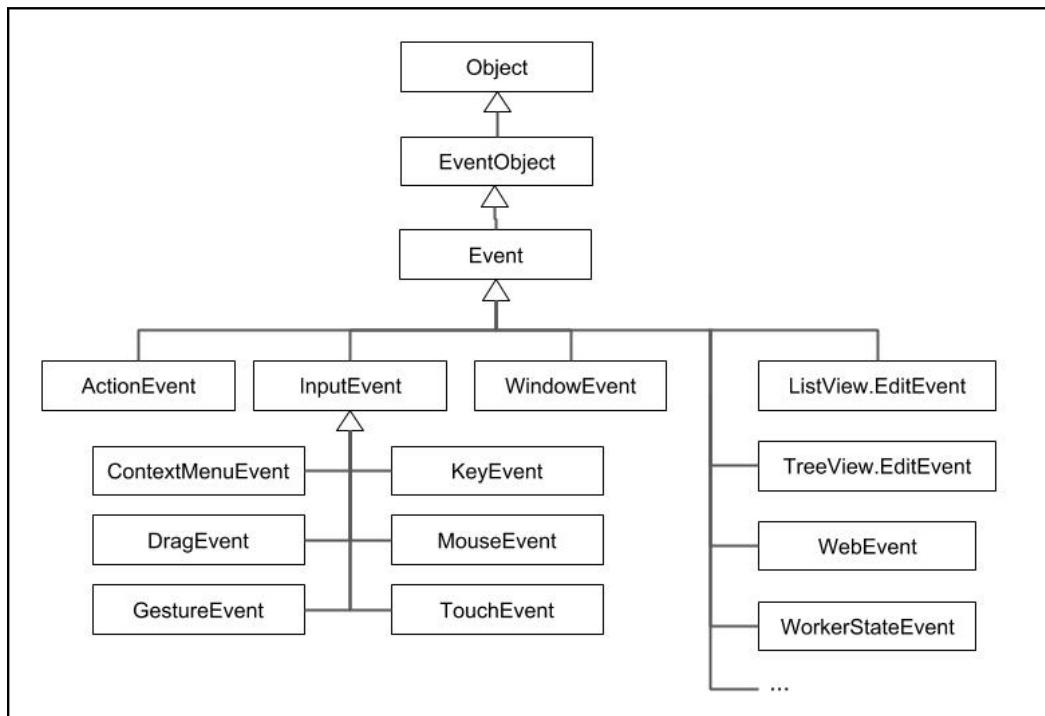


Figura 1.2: Diagrama de clases de eventos en JavaFX

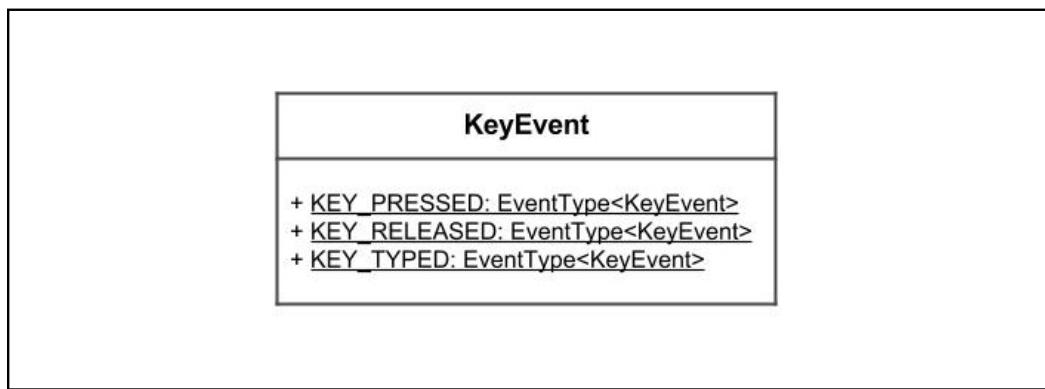


Figura 1.3: Diagrama de clase KeyEvent

Tabla 1.1: Relación Event - EventType

Subclase	Subtipos
MouseEvent	MOUSE_DETECTED, MOUSE_CLICKED, MOUSE_DRAGGED, MOUSE_MOVED, MOUSE_ENTERED, MOUSE_EXITED, MOUSE_MOVED, MOUSE_PRESSED y MOUSE_RELEASED
KeyEvent	KEY_PRESSED, KEY_RELEASED y KEY_TYPED
TouchEvent	TOUCH_MOVED, TOUCH_PRESSED, TOUCH_RELEASED y TOUCH_STATIONARY

## Tipos de Oyentes

Las clases oyentes representan objetos que pueden manipular los eventos que escuchan. JavaFX utiliza la interface `EventHandler` para el manejo de todos los tipos de eventos posibles, el código 1.1 muestra su definición.

Código 1.1: Definición de la interface EventHandler

```

1 public interface EventHandler<T extends Event> extends EventListener {
2
3     void handle(T event);
4
5 }
```

`EventHandler` es una interface paramétrizada, esto significa que su definición es aún más general que el de una interface normal, dando posibilidad al programador de especificar los tipos de retorno o argumentos de sus métodos mediante el parámetro que se encuentra entre los símbolos `< >`.

En la línea 1 del código 1.1, se indica que el parámetro `T` tiene que ser subclase de `Event`, y este parámetro modificará el tipo del argumento de la función `handle()` de la línea 3. Es el uso del parámetro de la interface lo que vuelve a `EventHandler` lo suficiente flexible y puede ser utilizada para escuchar y atender cualquier tipo de evento.

Estos son los cuatro modos de implementar un oyente con la interface `EventHandler`:

**Implementación externa:** Una clase independiente (clase que es ajena a cualquier otra) implementa la interface `EventHandler`, posteriormente se registra una nueva instancia del oyente en el objeto que origina los eventos. Vea el código 1.4.

**Implementación interna:** La interface `EventHandler` se implementa en alguna de las otras clases de la aplicación, una clase contenedora por ejemplo y posteriormente se registra la instancia de la clase que implementó la interface como oyente en el objeto que origina los eventos. Vea los códigos 1.5 y 1.6.

**Clase anónima:** Se registra directamente una instancia de la interface `EventHandler` implementando el método abstracto durante el instanciamiento. Vea el código 1.7.

**Lambda:** Se registra una función lambda (función anónima) definiendo su cuerpo durante el registro. Vea el código 1.8.

## Métodos de Registro de Oyentes

Toda subclase de `Node` tiene métodos de conveniencia para registrar los eventos más generales definidos en JavaFX, `setOnMouseClicked()`, `setOnKeyPressed()`, `setOnTouchMoved()`, `setOnRotate()`, entre muchos otros. El código 1.2 presenta, de forma general, una firma de tipo de los métodos para registrar oyentes de eventos. Las letras en negritas son ilustrativas.

Código 1.2: Firma de tipo de los métodos de la clase `Node` para registrar oyentes

```
public void setOnEventName(SubclassOfEvent event)
```

Adicionalmente `Node` tiene un método para registrar cualquier tipo de evento, su firma de tipo se muestra en el código 1.3.

Código 1.3: Firma de tipo del método `addEventHandler()`

```
public final <T extends Event> void addEventHandler(EventType<T> eventType,
                                                     EventHandler<? super T> eventHandler)
```

El primer parámetro del método indica el tipo de evento que va a estar escuchando, el segundo parámetro es el oyente.

Código 1.4: Implementación de oyente en clase externa

```
//Implementación del Oyente
public interface ButtonListener implements EventHandler<ActionEvent> {

    @Override
    public void handle(ActionEvent event) {
        ...
        System.out.println("Do_something");
        ...
    }
}

public class Layout extends BorderPane {

    public void someMethod() {
        Button btn = new Button("Click_Me!");
        btn.setOnAction(new ButtonListener()); //Registro del oyente
        ...
    }
}
```

Código 1.5: Implementación de un oyente en clase contenedora del origen de eventos

```
public class Layout extends BorderPane implements EventHandler<ActionEvent> {

    public void someMethod() {
        Button btn = new Button("Click_Me!");
        btn.setOnAction(this); //Registro del oyente
        ...
    }

    //Implementación del Oyente
    @Override
    public void handle(ActionEvent event) {
        System.out.println("Do_something!");
    }
}
```

Código 1.6: Implementación de un oyente en clase interna

```
public class Layout extends BorderPane {

    public void someMethod() {
        Button btn = new Button("Click_Me!");
        btn.setOnAction(new ButtonListener()); //Registro del oyente
        ...
    }

    class ButtonListener implements EventHandler<ActionEvent> {

        //Implementación del Oyente
        @Override
        public void handle(ActionEvent event) {
            System.out.println("Do_something!");
        }
    }
}
```

Código 1.7: Implementación de oyente con clase anónima

```
public class Layout extends BorderPane {

    public void someMethod() {
        Button btn = new Button("Click_Me!");
        //Registro del Oyente
        btn.setOnAction(new EventHandler<ActionEvent>() {

            //Implementación del Oyente
            @Override
            public void handle(ActionEvent event) {
                System.out.println("Do_something!");
            }
        });
        ...
    }
}
```

Código 1.8: Implementación de oyente con lambda

```
public class Layout extends BorderPane {  
  
    public void someMethod() {  
        Button btn = new Button("Click Me!");  
        //Registro del Oyente  
        btn.setOnAction(ActionEvent e) -> {  
            System.out.println("Do some cool stuff");  
        });  
        ...  
    }  
}
```

## Ejemplos Prácticos

### Ejemplo 1: Manejo de Eventos de Botón

En este ejemplo se manejan eventos de botón mediante un esquema de oyente con clase externa.

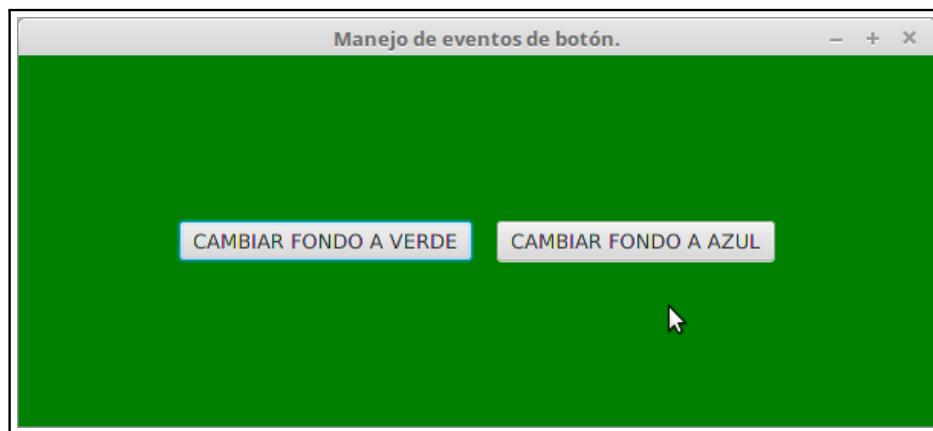


Figura 1.4: Aplicación con eventos de botón

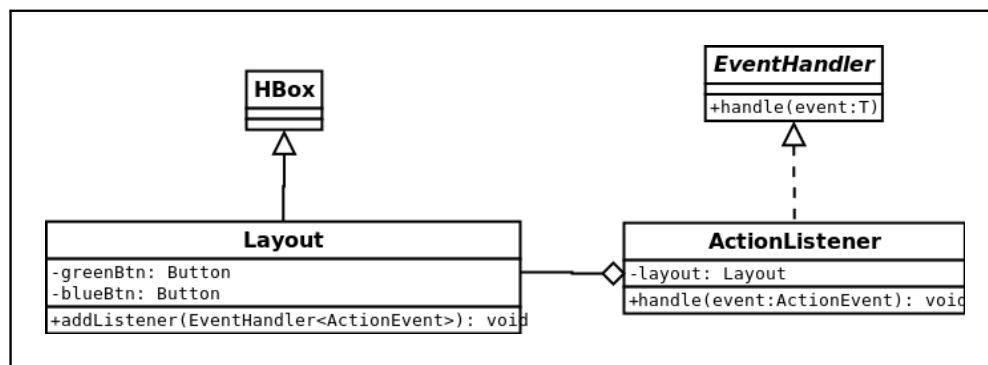


Figura 1.5: Diagrama de clases de la aplicación para el manejo de eventos de botón con oyente externo

Código 1.9: Clase principal de la aplicación

```
public class Main extends Application {  
  
    @Override  
    public void start(Stage primaryStage) {  
        Layout root = new Layout();  
        EventHandler<ActionEvent> listener = new ActionListener(root);  
        root.addListener(listener);  
  
        primaryStage.setScene(new Scene(root, 600, 400));  
        primaryStage.setTitle("Manejo de eventos de botón.");  
        primaryStage.show();  
    }  
  
    public static void main(String [] args) { launch(); }  
}
```

Código 1.10: Layout de la interfaz gráfica de la aplicación

```
public class Layout extends HBox {  
  
    private Button greenBtn;  
    private Button blueBtn;  
  
    public Layout() { init(); }  
  
    private void init() {  
        setAlignment(Pos.CENTER);  
        setSpacing(16);  
  
        greenBtn = new Button("CAMBIAR FONDO A VERDE");  
        blueBtn = new Button("CAMBIAR FONDO A AZUL");  
  
        greenBtn.setId("green-btn");  
        blueBtn.setId("blue-btn");  
  
        getChildren().addAll(greenBtn, blueBtn);  
    }  
  
    public void addListener(EventHandler<ActionEvent> listener) {  
        greenBtn.setOnAction(listener);  
        blueBtn.setOnAction(listener);  
    }  
}
```

Código 1.11: Clase que implementa EventHandler

```

public class ActionListener implements EventHandler<ActionEvent> {

    private final Pane layout;

    public ActionListener(Pane layout) {
        this.layout = layout;
    }

    @Override
    public void handle(ActionEvent event) {
        String sourceId = ((Node) event.getSource()).getId();

        switch(sourceId) {
            case "green-btn":
                layout.setBackground(new Background(
                    new BackgroundFill(Color.GREEN, null, null)));
                break;

            case "blue-btn":
                layout.setBackground(new Background(
                    new BackgroundFill(Color.BLUE, null, null)));
                break;
        }
    }
}

```

**Ejemplo 2: Oyente Interno - Implementación de la Interface en Contenedor**

En este ejemplo se obtiene una aplicación con el mismo comportamiento que la del ejemplo anterior pero diferente en implementación de oyentes. Este oyente se implementa en la clase contenedora de los orígenes de eventos (los botones).

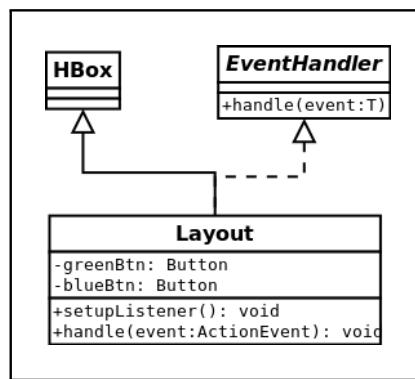


Figura 1.6: Diagrama de clases de la aplicación para el manejo de eventos de botón con oyente interno

Código 1.12: Clase principal de la aplicación

```
public class Main extends Application {

    @Override
    public void start(Stage primaryStage) {
        Layout root = new Layout();

        primaryStage.setScene(new Scene(root, 600, 400));
        primaryStage.setTitle("Manejo_de_eventos_de_botón.");
        primaryStage.show();
    }

    public static void main(String [] args) { launch(); }
}
```

Código 1.13: Oyente implementado en la clase Layout

```
public class Layout extends HBox implements EventHandler<ActionEvent> {

    private Button greenBtn;
    private Button blueBtn;

    public Layout() { init(); setupListener(); }

    private void init() {
        //Mismo código que en el ejemplo 1
    }

    private void setupListener() {
        greenBtn.setOnAction(this);
        blueBtn.setOnAction(this);
    }

    @Override
    public void handle(ActionEvent event) {
        String sourceId = ((Node) event.getSource()).getId();

        switch (sourceId) {
            case "green-btn":
                this.setBackground(new Background(
                    new BackgroundFill(Color.GREEN, null, null)));
                break;

            case "blue-btn":
                this.setBackground(new Background(
                    new BackgroundFill(Color.BLUE, null, null)));
                break;
        }
    }
}
```

**Ejemplo 3: Oyente Interno - Implementación de la Interface en Clase interna**

Similar al ejemplo 2, en este ejemplo se define el oyente dentro de la misma clase contenedora de los orígenes de eventos pero se implementa en una clase interna. En el diagrama de la figura 1.7 se indica la clase interna mediante el conector con punta circular. El conector con punta romboidal indica que dentro de la clase `Layout` se hace uso de la clase `ActionListener`.

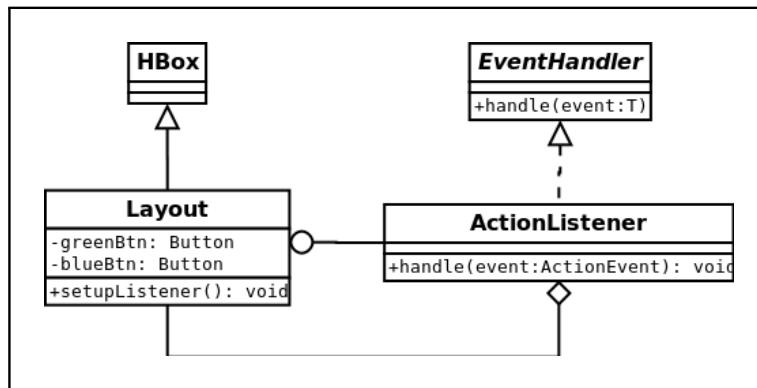


Figura 1.7: Diagrama de clases de la aplicación para el manejo de eventos de botón con oyente en clase interno

Código 1.14: Clase principal de la aplicación

```

public class Main extends Application {
    @Override
    public void start(Stage primaryStage) {
        Layout root = new Layout();

        primaryStage.setScene(new Scene(root, 600, 400));
        primaryStage.setTitle("Manejo_de_eventos_de_botón.");
        primaryStage.show();
    }

    public static void main(String [] args) { launch(); }
}
    
```

Código 1.15: Clase interna como oyente

```
public class Layout extends HBox {

    private Button greenBtn;
    private Button blueBtn;

    public Layout() { init(); setupListener(); }

    private void init() {
        //Mismo código que en el ejemplo 1
    }

    private void setupListener() {
        ActionListener listener = new ActionListener();
        greenBtn.setOnAction(listener);
        blueBtn.setOnAction(listener);
    }

    class ActionListener implements EventHandler<ActionEvent> {

        @Override
        public void handle(ActionEvent event) {
            String sourceId = ((Node) event.getSource()).getId();

            switch (sourceId) {
                case "green-btn":
                    Layout.this.setBackground(new Background(
                        new BackgroundFill(Color.GREEN, null, null)));
                    break;

                case "blue-btn":
                    Layout.this.setBackground(new Background(
                        new BackgroundFill(Color.BLUE, null, null)));
                    break;
            }
        }
    }
}
```

**Ejemplo 4: Implementación de oyente en clase anónima**

Este método también se puede considerar una implementación interna pero se hace una distinción debido a que la implementación carece de una clase receptora de la interface.

Código 1.16: Clase principal de la aplicación

```
public class Main extends Application {

    @Override
    public void start(Stage primaryStage) {
        Layout root = new Layout();

        primaryStage.setScene(new Scene(root, 600, 400));
        primaryStage.setTitle("Manejo de eventos de botón.");
        primaryStage.show();
    }

    public static void main(String [] args) { launch(); }
}
```

Código 1.17: Clase contenedora con oyente anónimo implementado

```
public class Layout extends HBox {

    private Button greenBtn;
    private Button blueBtn;

    public Layout() { init(); setupListener(); }

    private void init() {
        //Mismo código que en el ejemplo 1
    }

    private void setupListener() {
        EventHandler<ActionEvent> listener = new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent event) {
                String sourceId = ((Node) event.getSource()).getId();

                switch (sourceId) {
                    case "green-btn":
                        Layout.this.setBackground(new Background(
                            new BackgroundFill(Color.GREEN, null, null)));
                        break;

                    case "blue-btn":
                        Layout.this.setBackground(new Background(
                            new BackgroundFill(Color.BLUE, null, null)));
                        break;
                }
            }
        };
        greenBtn.setOnAction(listener);
        blueBtn.setOnAction(listener);
    }
}
```

**Ejemplo 5: Lambda como Oyente**

Las interfaces que poseen un sólo método son conocidas como interfaces funcionales, debido a que su definición como clases anónimas pueden ser sustituidas por funciones lambda.

Código 1.18: Clase principal de la aplicación

```
public class Main extends Application {

    @Override
    public void start(Stage primaryStage) {
        Layout root = new Layout();

        primaryStage.setScene(new Scene(root, 600, 400));
        primaryStage.setTitle("Manejo de eventos de botón.");
        primaryStage.show();
    }

    public static void main(String [] args) { launch(); }
}
```

Código 1.19: Clase contenedora con oyente anónimo implementado

```
public class Layout extends HBox {

    private Button greenBtn;
    private Button blueBtn;

    public Layout() { init(); setupListener(); }

    private void init() {
        //Mismo código que en el ejemplo 1
    }

    private void setupListener() {
        EventHandler<ActionEvent> listener = (ActionEvent event) -> {
            String sourceId = ((Node) event.getSource()).getId();

            switch (sourceId) {
                case "green-btn":
                    Layout.this.setBackground(new Background(
                        new BackgroundFill(Color.GREEN, null, null)));
                    break;

                case "blue-btn":
                    Layout.this.setBackground(new Background(
                        new BackgroundFill(Color.BLUE, null, null)));
                    break;
            };
            greenBtn.setOnAction(listener);
            blueBtn.setOnAction(listener);
        }
    }
}
```

**Ejemplo Extra: FXML y Controladores**

El esquema de construcción de interfaces mediante archivos FXML permite otra forma de atender los eventos de los componentes de la GUI.

FXML es un lenguaje declarativo, no de programación, y es utilizado para describir la interface de usuario: los componentes que la conforman, su estructura y orden. Para definir el comportamiento de la aplicación cuando se produce algún eventos desde la GUI se utiliza código Java.

En FXML se pueden definir uno o más controladores (oyentes) en las distintas secciones de la GUI. La vinculación de un controlador con la sección de GUI que controlará se hace mediante el atributo `fx:controller`.

A continuación se presentan códigos de ejemplo para realizar la misma aplicación de los ejemplos anteriores utilizando FXML.

Código 1.20: Clase principal donde se construye la GUI desde el archivo FXML

```
package fxml;

//Se omiten imports
import javafx.fxml.FXMLLoader;

public class Main extends Application {

    @Override
    public void start(Stage stage) throws Exception {
        Parent root = FXMLLoader.load(getClass().getResource("layout.fxml"));

        Scene scene = new Scene(root);
        stage.setScene(scene);
        stage.show();
    }

    public static void main(String[] args) { launch(args); }
}
```

Código 1.21: Archivo FXML que define la GUI de la aplicación

```
<?xml version="1.0" encoding="UTF-8"?>
<?import javafx.scene.control.Button?>
<?import javafx.scene.layout.HBox??>

<HBox xmlns="http://javafx.com/javafx/8.0.111" xmlns:fx="http://javafx.com/fxml/1"
       maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-Infinity"
       prefHeight="331.0" prefWidth="557.0" spacing="16.0" alignment="CENTER"
       fx:id="container" fx:controller="fxml.Controller">
    <children>
        <Button mnemonicParsing="false" onAction="#changeToGreen" text="CAMBIAR_FONDO_A_VERDE" />
        <Button mnemonicParsing="false" onAction="#changeToBlue" text="CAMBIAR_FONDO_A_AZUL" />
    </children>
</HBox>
```

Código 1.22: Clase controladora de los eventos de la GUI

```
1 package fxml;
2
3 //Se omiten imports
4 import javafx.fxml.FXML;
5
6 public class Controller {
7
8     @FXML
9     private HBox container;
10
11     @FXML
12     private void changeToGreen(ActionEvent event) {
13         container.setBackground(new Background(
14             new BackgroundFill(Color.GREEN, null, null)));
15     }
16
17     @FXML
18     private void changeToBlue(ActionEvent event) {
19         container.setBackground(new Background(
20             new BackgroundFill(Color.BLUE, null, null)));
21     }
22 }
23 }
```

En el código 1.22 se utiliza la anotación `@FXML` para dos propósitos. En la línea 8 vincula el atributo que le sucede con un componente de la interface definido en el archivo FXML, el pegamento que los une son el nombre del atributo de la clase y del atributo `fx:id` del elemento `HBox` en el archivo FXML. La segunda función de la anotación es "exponer" los métodos de las líneas 12 y 18 del controlado en el archivo FXML permitiendo su asociación como oyentes de los eventos de los botones.