.UBAfiuba
FACULTAD DE INGENIERÍA

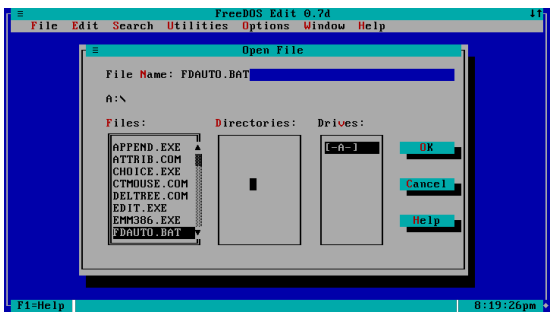
Desarrollo de Aplicaciones con Interfaces Gráficas

TB025 - Paradigmas de Programación

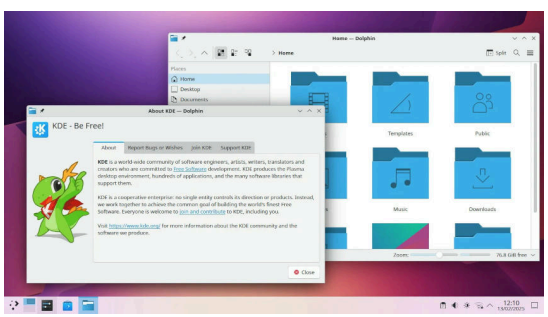
Curso Essaya

Interfaces de usuario

CLI



TUI



GUI

REPL vs Event loop

Las interfaces de línea de comandos (CLI) suelen ser implementadas mediante un algoritmo simple: **REPL**.

```
function main
    initialize()
    while command != quit
        show_prompt()
        command := read_line()
        result := process_command(command)
        show_result(result)
    end while
end function
```

Dado que las interfaces de usuario deben lidiar con múltiples entradas y salidas (mouse, teclado, monitor, etc.), la implementación más usual es mediante un **event loop**.

```
function main
    initialize()
    while event != quit
        event := get_next_event()
        process_event(event)
    end while
end function
```

Los **frameworks GUI** suelen abstraer el event loop; el cliente solo debe escribir el código que procesa los eventos.

Gráficos, Widgets y aplicaciones

Existe una gran cantidad de opciones de librerías y frameworks gráficos, cada una con sus propias características, ventajas y desventajas. Para elegir una hay que tener en cuenta requerimientos como la **plataforma**, **lenguaje de programación**, **licencia**, etc.

Además, considerar el **nivel de abstracción** deseado. Algunos ejemplos:

Librerías gráficas (bajo nivel): OpenGL, Vulkan, DirectX

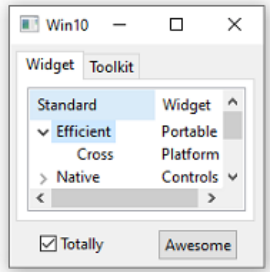
Se encargan de dibujar píxeles en la pantalla, pero no proveen componentes de interfaz de usuario.

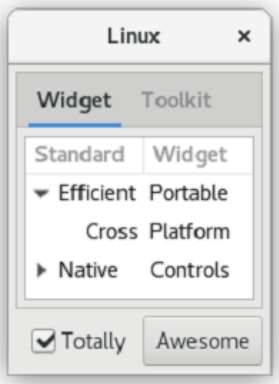
Widget toolkits: GTK, Qt, Tk, wxWidgets, WinUI, Cocoa

Facilitan la creación de interfaces gráficas, con **ventanas**, y **widgets/controles** (botones, menús, etc.).

Application frameworks (alto nivel): Electron, Tauri, Flutter, JavaFX

Proveen un entorno completo para desarrollar aplicaciones, incluyendo en algunos casos soporte de hardware específico como la cámara, micrófono, GPS, etc. El **event loop** suele estar abstraido en esta capa.



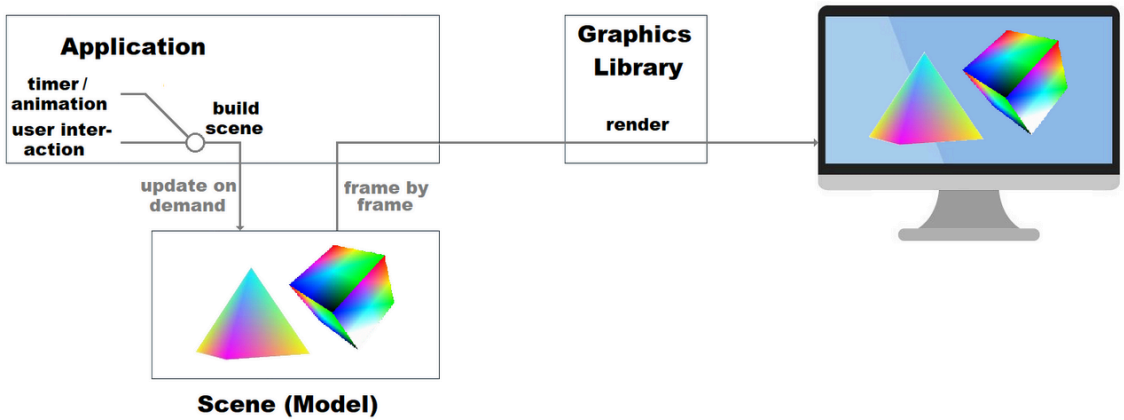


Modo inmediato vs modo retenido

Modo inmediato :

- La escena se mantiene en la memoria de la aplicación.
- La aplicación es responsable de invocar a la librería gráfica para redibujar la pantalla.

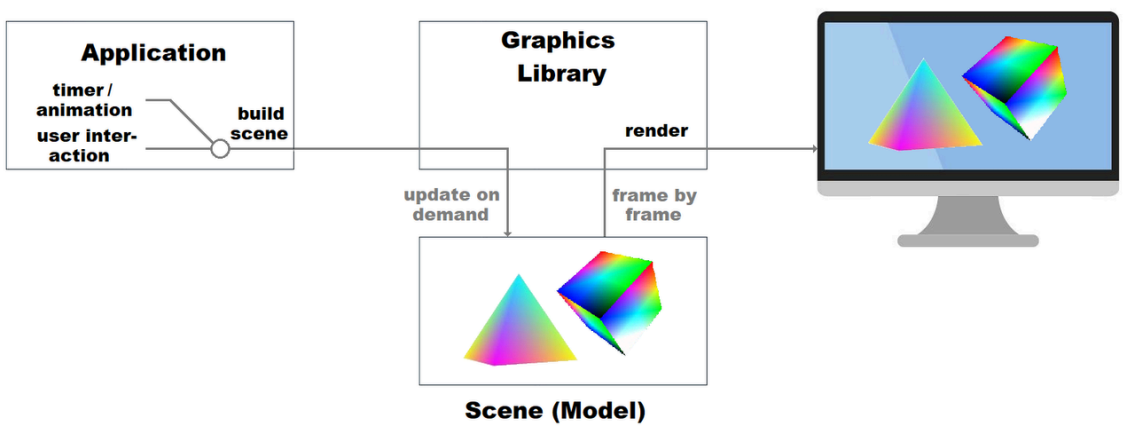
Las librerías gráficas de bajo nivel suelen ser de modo inmediato (ej: OpenGL).



Modo retenido :

- La escena se mantiene en la memoria de la librería gráfica.
- La librería gráfica se encarga de redibujar la pantalla en caso de ser necesario.

Los frameworks de alto nivel suelen ser de modo retenido (ej: JavaFX).



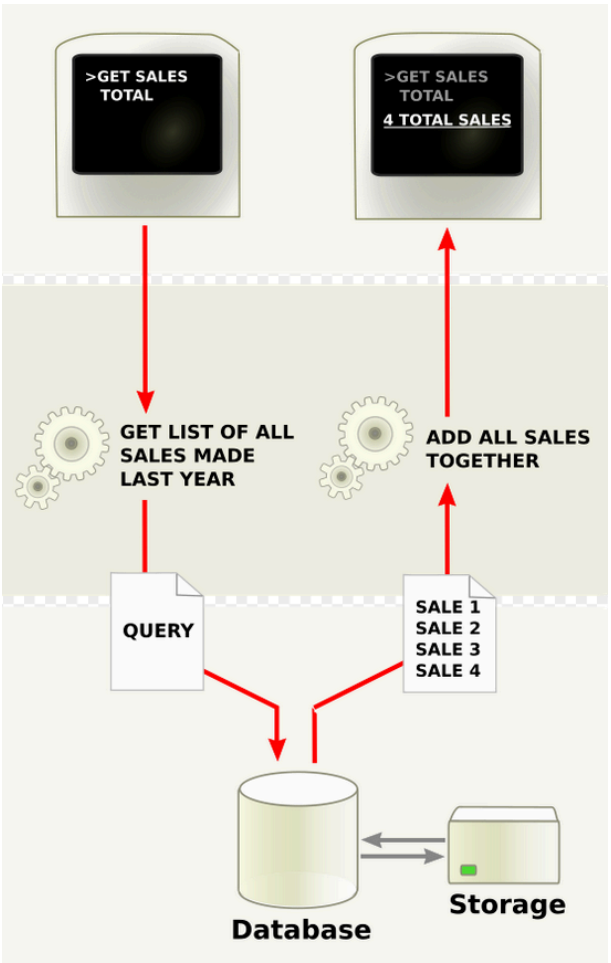
Capas de abstracción

Siguiendo el principio **SoC**, las aplicaciones con UI suelen tener al menos 2 o 3 capas de abstracción:

Presentación: Se encarga de la **interacción con el usuario**.

Lógica de negocio: Se encarga de modelar el **estado** de la aplicación y las **operaciones** que pueden realizarse sobre él.

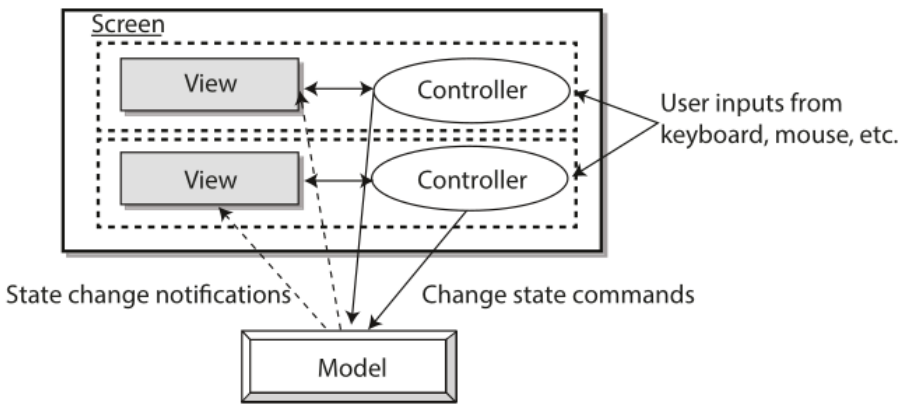
Persistencia: Se encarga de **almacenar** el estado de la aplicación de forma permanente, por ejemplo en una base de datos o un archivo.

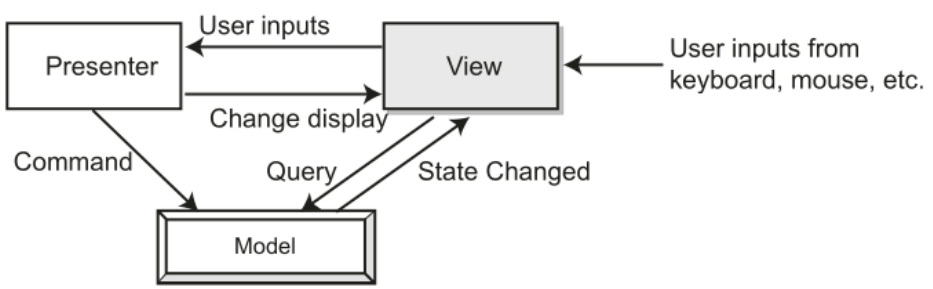


El patrón MVC

Muchos frameworks se basan en la arquitectura **MVC** (Model-View-Controller), para desacoplar la lógica de negocio (Modelo), la representación gráfica (Vista) y la interacción del usuario (Controlador). Este patrón fue creado para desarrollar aplicaciones GUI en Smalltalk-80.

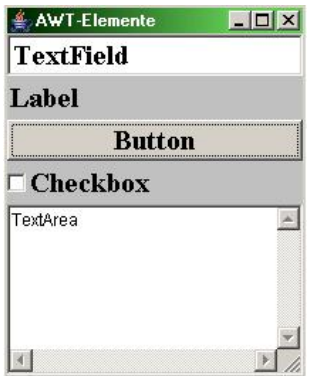
Los frameworks modernos adoptan diferentes variantes de MVC. Por ejemplo, en el patrón **MVP** (Modelo-Vista-Presentación) los *widgets* de la Vista son responsables de interceptar el *input* del usuario y enviarlo a la Presentación.



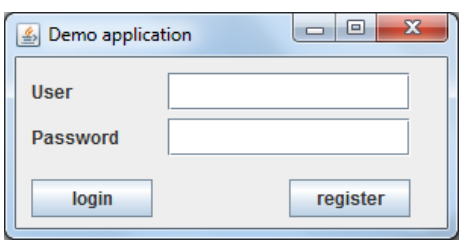


Desarrollo de aplicaciones con GUI en Java

AWT (Abstract Window Toolkit) : Es el *widget toolkit* original de Java, creado en 1995. Tiene la particularidad de que los componentes son nativos del sistema operativo: se ven y se comportan como los controles del sistema operativo en el que se ejecutan. Esto puede ser una ventaja en términos de apariencia y usabilidad, pero también puede ser una desventaja si se busca una apariencia consistente en diferentes plataformas.



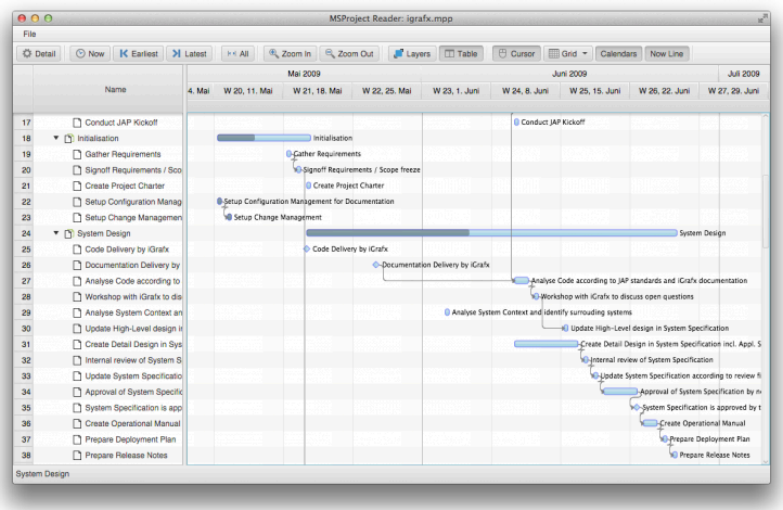
Swing : Reemplaza a AWT como el *widget toolkit* estándar de Java SE en 1998. Fue diseñado para ser más sofisticado y flexible que AWT. Permite modificar el estilo visual de los componentes, y es completamente independiente del sistema operativo: los componentes se ven iguales en todas las plataformas.



JavaFX

Es un framework para crear aplicaciones sobre la plataforma Java.

Fue lanzado en 2008 por Sun Microsystems para reemplazar a Swing como la biblioteca GUI estándar para Java SE, pero actualmente ya no forma parte de las ediciones estándar, mientras que Swing y AWT permanecen incluidas. En 2018 Oracle hizo que JavaFX fuera parte del proyecto OpenJDK, bajo el nombre **OpenJFX**.



Entre otras, posee las siguientes funcionalidades:

- Gráficos 2D y 3D acelerados
- Controles, diseños y cuadros/tablas de la GUI
- Soporte de audio y video
- Efectos y animaciones
- Definición de la estructura GUI con FXML
- Personalización de los estilos con CSS

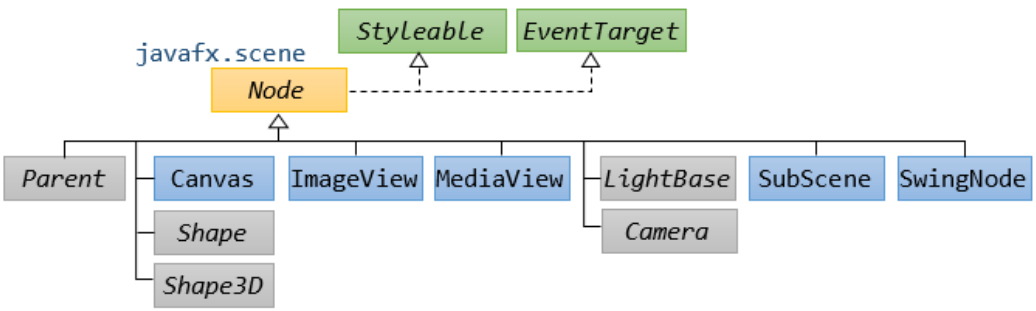
Para proporcionar el máximo rendimiento, JavaFX utiliza diferentes motores de renderizado nativos según la plataforma en la que se ejecuta. En Windows, por ejemplo, se utiliza Direct3D, mientras que en la mayoría de los demás sistemas se utiliza OpenGL.

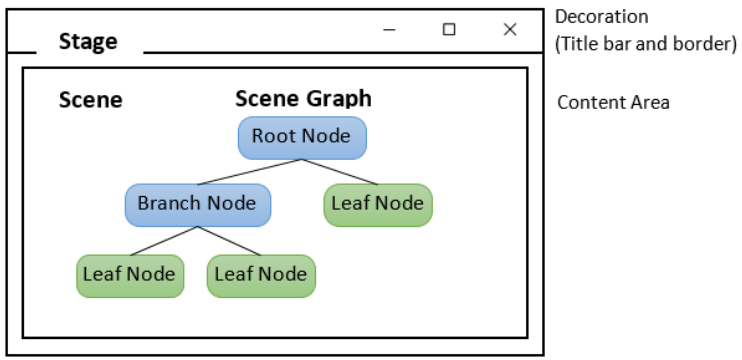
Hola Mundo

```
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.text.Font;
import javafx.scene.text.Text;
import javafx.stage.Stage;

public class HelloWorld extends Application {

    @Override public void start(Stage stage) {
        Text text = new Text(10, 40, "Hello World!");
        text.setFont(new Font(40));
        Scene scene = new Scene(new Group(text));

        stage.setTitle("Welcome to JavaFX!");
        stage.setScene(scene);
        stage.sizeToScene();
        stage.show();
    }

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

Scene Graph

El **Scene Graph** es la estructura de datos y API asociada de JavaFX para manipular la escena. Es muy similar en funcionamiento al **DOM**: los elementos forman un árbol y los eventos se propagan a través de él.

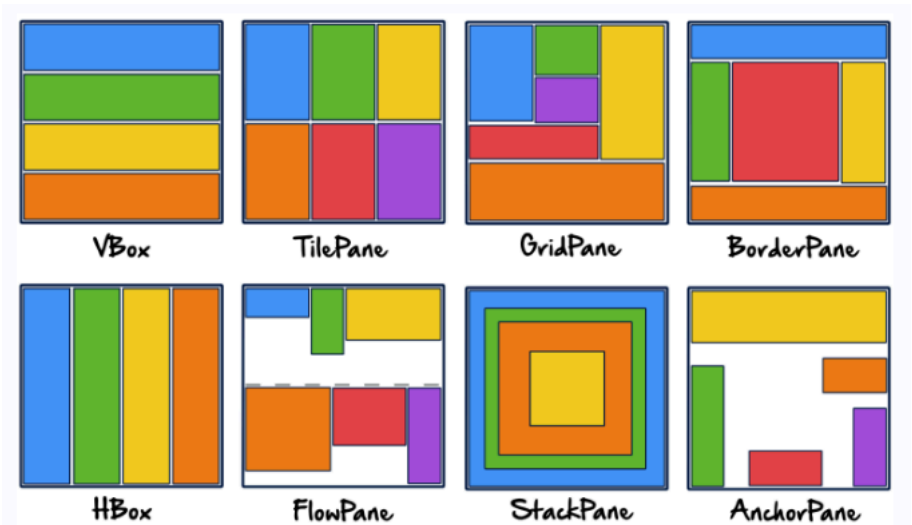


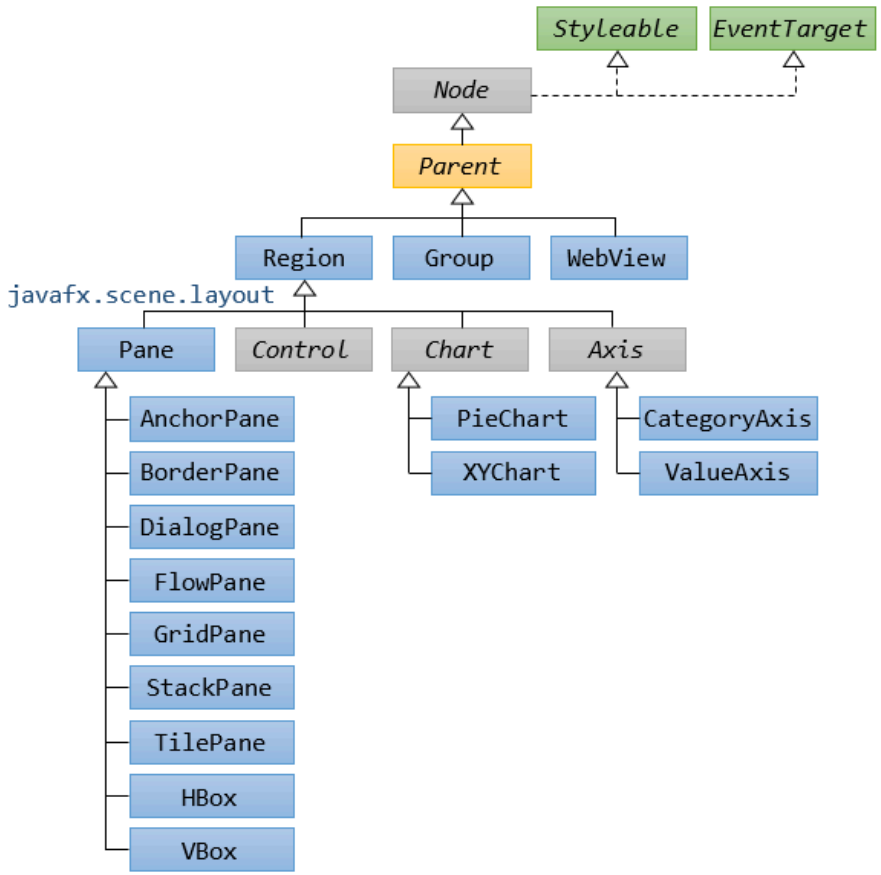


Stage: representa una ventana en la pantalla. Contiene un **Scene**.
Scene: representa el contenido de la ventana. Contiene el **Scene Graph**, mediante una referencia al nodo raíz.
Node: Clase base para todos los nodos del **Scene Graph**.
Parent: Clase base para los nodos que pueden tener hijos.

Scene Graph: Layout

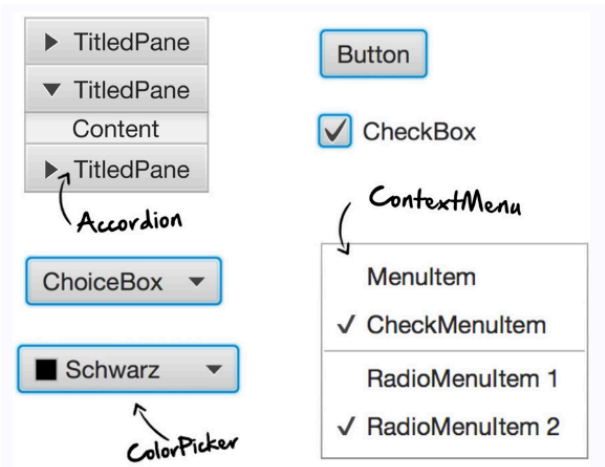
Estas clases permiten agrupar nodos para posicionar y redimensionar con diferentes estrategias.

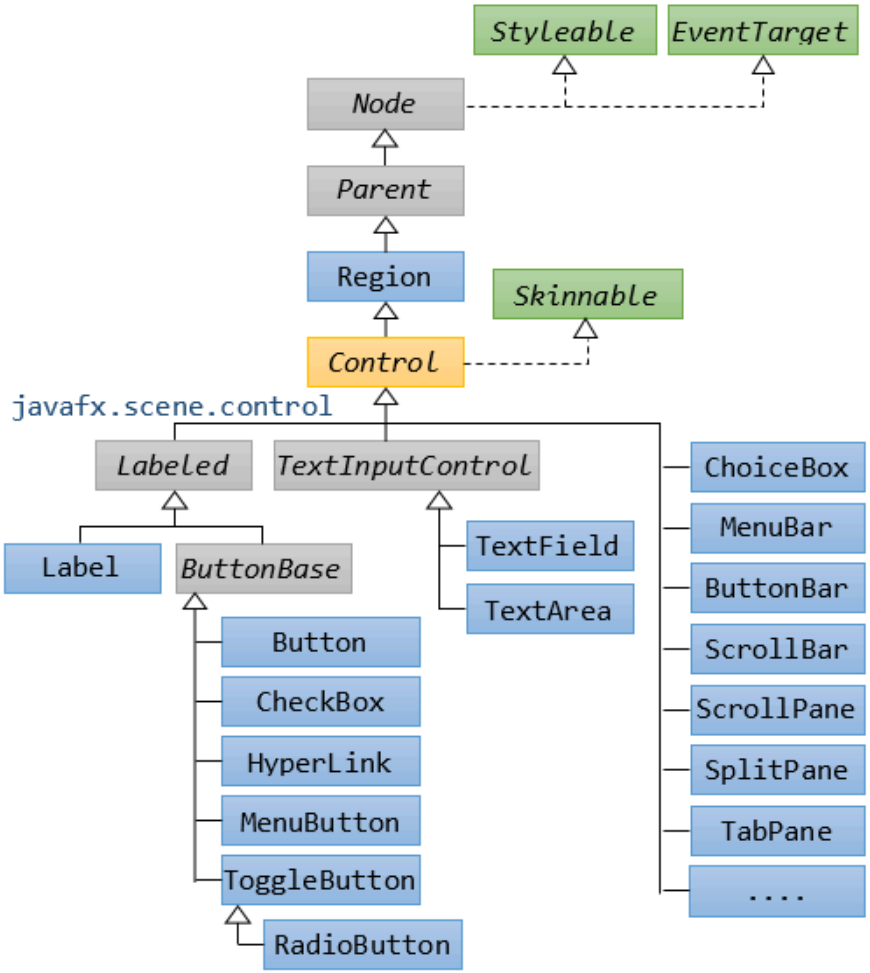




Scene Graph: Controles

Son los componentes interactivos de la interfaz de usuario, como botones, etiquetas, campos de texto, etc.

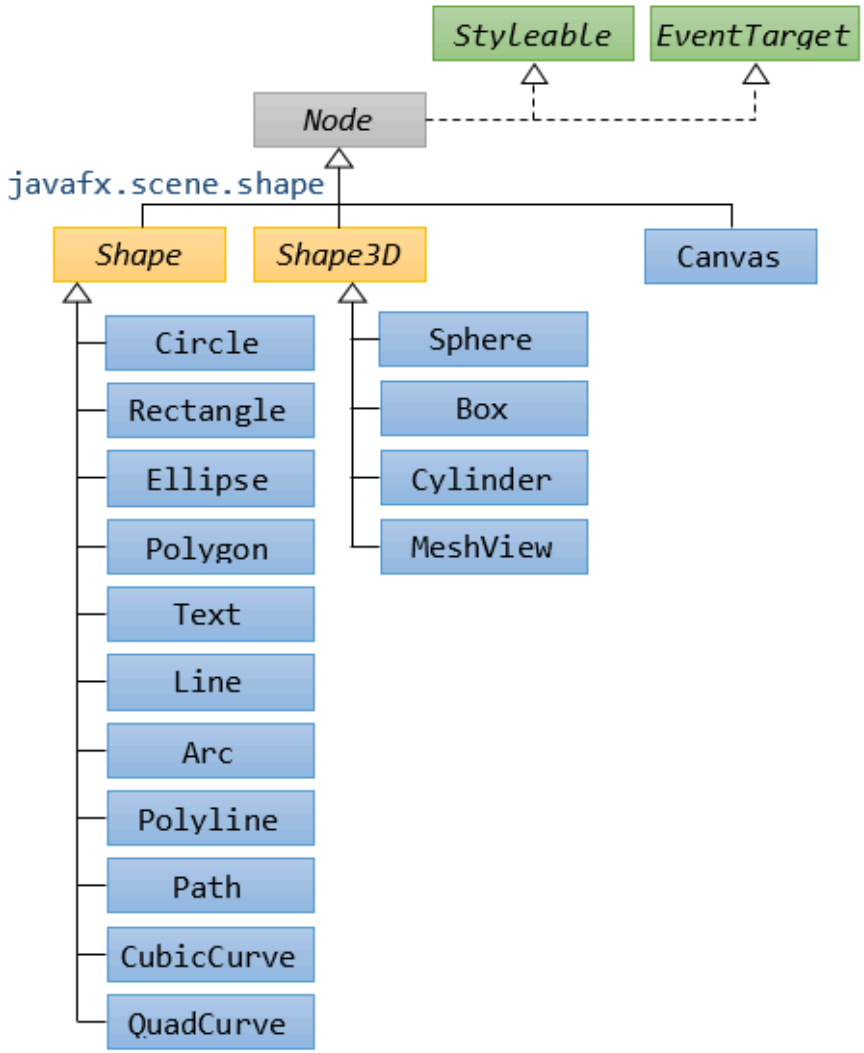




Scene Graph: Shapes

Permiten dibujar formas geométricas básicas como líneas, círculos, rectángulos, polígonos, etc.

El **Canvas** es más eficiente para dibujar gráficos complejos, ya que permite dibujar directamente en un área rectangular, sin necesidad de crear un nodo para cada forma. Es útil para gráficos dinámicos o animaciones.



Properties & Bindings

Las clases del paquete `javafx.beans.property` ofrecen interfaces e implementaciones de **properties**, que encapsulan valores que pueden ser **observados** y **enlazados** (*bindings*).

```
IntegerProperty num = new SimpleIntegerProperty(42);
num.addListener((_, viejo, nuevo) -> {
    System.out.printf("El número cambió de %s a %s\n", viejo, nuevo);
});
num.set(num.get() + 1);
```

```
IntegerProperty a = new SimpleIntegerProperty(3);
IntegerProperty b = new SimpleIntegerProperty(5);
IntegerProperty c = new SimpleIntegerProperty();
NumberBinding suma = a.add(b);
c.bind(suma); // binding: c = a + b

System.out.println(c.getValue()); // 8
a.set(10);
System.out.println(c.getValue()); // 15
```

Properties & Bindings (cont.)

Ejemplo de uso de **bindings** para centrar un elemento en la ventana:

```
Circle c = new Circle(30, Color.BLACK);

Group root = new Group(c);
Scene scene = new Scene(root, 150, 150);

c.centerXProperty().bind(scene.widthProperty().divide(2));
c.centerYProperty().bind(scene.heightProperty().divide(2));

stage.setScene(scene);
stage.show();
```

Properties & Bindings (cont.)

El mecanismo de *bindings* también se puede usar para automatizar la actualización de la *vista* a partir de los cambios del *modelo*:

```
// modelo
class Persona {
    public final StringProperty nombre = new SimpleStringProperty("Juan");
    public final IntegerProperty edad = new SimpleIntegerProperty(20);
}
```

```
// vista
Persona p = new Persona();

TextField nombre = new TextField();
nombre.textProperty().bindBidirectional(p.nombre);

Button incrementar = new Button("incrementar");
incrementar.setOnAction(_ -> {
    p.edad.set(p.edad.get() + 1);
});

Label label = new Label();
label.textProperty().bind(Bindings.format("%s tiene %d años", p.nombre, p.edad));
```

Nota: esto requiere que el modelo dependa de JavaFX.

FXML

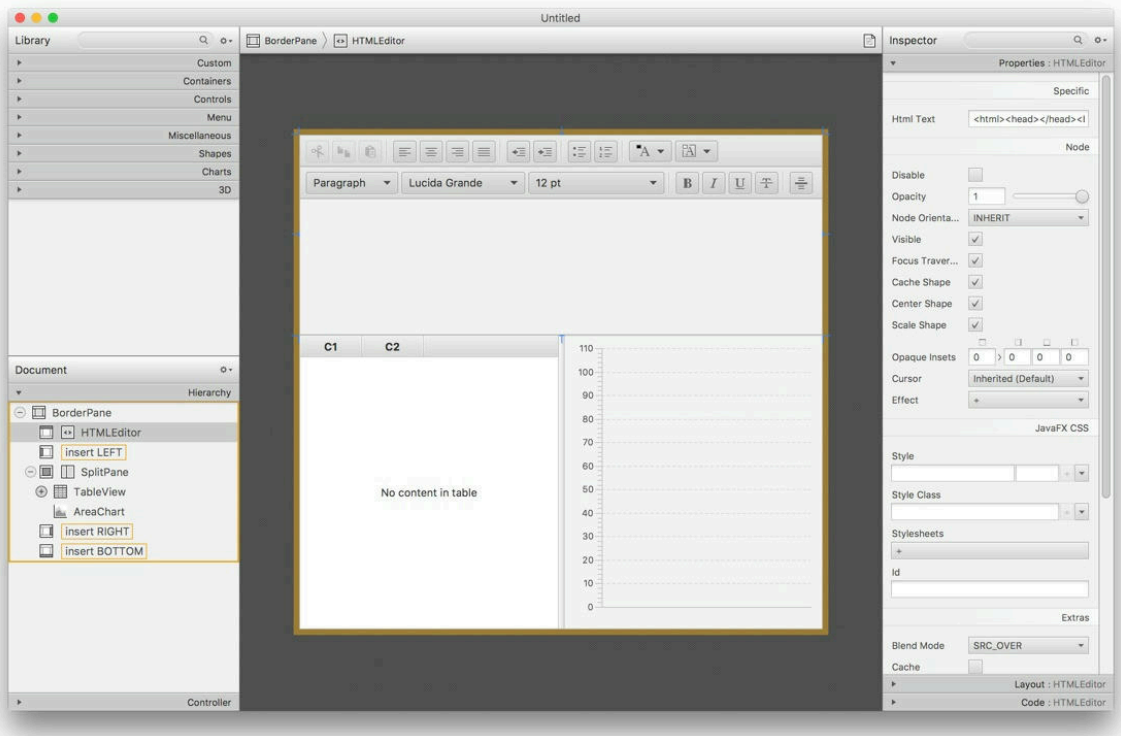
Permite definir la estructura del Scene Graph en un archivo XML, separando la **presentación** de la **lógica**. Es similar a HTML, pero con etiquetas específicas de JavaFX.

```
<VBox>
    <children>
        <TextField fx:id="textField" />
        <Button fx:id="btn" text="Click Me!" />
    </children>
</VBox>
```

```
public class CustomControl extends VBox {
    @FXML private TextField textField;
    @FXML private Button btn;

    public CustomControl() {
        FXMLLoader fxmlLoader = new FXMLLoader(getClass().getResource("custom_control.fxml"));
        fxmlLoader.setRoot(this);
        fxmlLoader.setController(this);
        fxmlLoader.load();
    }
}
```

Scene Builder

Es una herramienta que permite diseñar interfaces gráficas de JavaFX de forma visual, arrastrando y soltando componentes. Genera el código FXML correspondiente automáticamente.



www.**ingenieria**.uba.ar

/ingenieriauba

/FIUBAoficial