UNIVERSIDAD DE GRANADA

Periféricos y Dispositivos de Interfaz Humana

Curso 2023

Proyecto de Teoría:

Proyecto sobre librerías/paquetes/módulos de interfaces, en distintos lenguajes de programación

---

Amador Carmona Méndez

Manuel Zafra Mota

Índice:

Introducción

En el mundo de la programación, las interfaces desempeñan un papel crucial en el desarrollo de software. Estas abstracciones permiten a los programadores diseñar y construir sistemas modulares, reutilizables y de fácil mantenimiento. Al utilizar librerías de interfaces en diferentes lenguajes de programación, como Python y Java, los desarrolladores tienen a su disposición un conjunto de herramientas poderosas para crear aplicaciones con interfaces gráficas intuitivas y atractivas.

En este trabajo, exploramos las librerías de interfaces más populares en Python y Java, dos de los lenguajes de programación más utilizados en la actualidad. Analizaremos las características y examinaremos ejemplos prácticos de su implementación en diversos contextos.

Librerías en Python

Las librerías de interfaces en Python proporcionan una interfaz de programación de aplicaciones (API) que permite a los desarrolladores interactuar con el SO y crear aplicaciones con una interfaz gráfica de usuario.

Estas librerías actúan como envoltorios alrededor de las funciones y características ofrecidas por el SO en relación con la creación y gestión de ventanas, widgets y eventos. Utilizan una combinación de código Python y código nativo del SO (escrito en C/C++ o en el lenguaje de programación nativo del SO) para lograr esta interacción. Algunas de las librerías de interfaces más populares en Python incluyen Tkinter, PyQt5, wxPython y Kivy.

Tkinter:

Principales características

1. Creación de ventanas: Tkinter permite crear ventanas principales y secundarias en las que se mostrará la interfaz gráfica, a las cuales se les puede modificar el tamaño, título y otras propiedades.
2. Diseño y posicionamiento de widgets: Tkinter proporciona diversos widgets, como botones, etiquetas, campos de entrada, cuadros de texto, listas desplegables, entre otros. Estos widgets se pueden colocar y organizar en la ventana utilizando técnicas de posicionamiento, como el uso de cuadrículas, empaquetado o posicionamiento absoluto.
3. Manejo de eventos: Permite asociar eventos, como clics de ratón o pulsaciones de teclas, con funciones específicas. Esto permite controlar y responder a las interacciones del usuario con la interfaz gráfica.
4. Personalización de apariencia: Tkinter proporciona opciones para personalizar la apariencia de los widgets y ventanas. Se pueden cambiar los colores, fuentes, tamaños, estilos y otros aspectos visuales para adaptar la interfaz gráfica al diseño deseado.

5. Cuadros de diálogo: Incluye una variedad de cuadros de diálogo predefinidos, como cuadros de mensaje, cuadros de entrada, cuadros de archivo, etc. Estos cuadros de diálogo facilitan la interacción con el usuario para obtener información o mostrar mensajes.
6. Control de diseño y geometría: Proporciona diferentes opciones para controlar la forma en que los widgets se ajustan y redimensionan en la ventana cuando ésta cambia de tamaño. Se pueden utilizar técnicas de gestión de geometría, como el uso de marcos, barras de desplazamiento y otras herramientas de diseño, para asi obtener una aplicación adaptable.
7. Integración de imágenes y gráficos: Permite cargar y mostrar imágenes en la interfaz gráfica, lo que resulta útil para presentar gráficos, iconos o cualquier otro elemento visual. También es posible dibujar formas geométricas y líneas utilizando métodos proporcionados por la librería.
8. Interacción con otras bibliotecas: Permite la combinación con otras bibliotecas y módulos de Python para ampliar su funcionalidad. Por ejemplo, se puede utilizar junto con la biblioteca *Matplotlib* para mostrar gráficos avanzados o con la biblioteca *NumPy* para la manipulación numérica.

Principales clases y funciones

- ➔ Tk: Es la clase principal que representa la ventana de la aplicación. Se utiliza para crear y gestionar la ventana principal de la interfaz gráfica.
- ➔ Frame: Es un contenedor rectangular que se utiliza para agrupar y organizar otros componentes.
- ➔ Label: Muestra texto o imágenes en la ventana.
- ➔ Button: Representa un botón que se puede presionar para activar una acción.
- ➔ Entry: Proporciona un campo de texto en el que el usuario puede ingresar y editar texto.
- ➔ Checkbutton: Muestra una casilla de verificación que puede ser seleccionada o deseleccionada.
- ➔ Radiobutton: Es similar a un Checkbutton, pero permite al usuario seleccionar una opción de un conjunto de opciones mutuamente excluyentes.
- ➔ Listbox: Muestra una lista de elementos entre los que el usuario puede realizar selecciones.
- ➔ Scrollbar: Proporciona una barra de desplazamiento que se puede usar con otros componentes, como Listbox, para permitir desplazarse por contenido adicional.
- ➔ Canvas: Ofrece un área de dibujo en la que se pueden representar gráficos, imágenes y formas geométricas.
- ➔ Menu: Permite crear menús y submenús en la barra de menú de la ventana principal.
- ➔ MessageBox: Muestra un cuadro de diálogo con un mensaje y botones para aceptar o cancelar.
- ➔ FileDialog: Proporciona un diálogo que permite al usuario seleccionar archivos o directorios.
- ➔ Spinbox: Permite al usuario seleccionar un valor numérico de un rango utilizando botones de aumento y disminución.

Ejemplo de código:

Python

```
import tkinter as tk
class ExampleApp(tk.Tk):

    def __init__(self):
        tk.Tk.__init__(self)
        self.title('Ejemplo de Tkinter con diseño responsive')

        # Crear un contenedor principal
        container = tk.Frame(self)
        container.pack(side='top', fill='both', expand=True)
        container.grid_rowconfigure(0, weight=1)
        container.grid_columnconfigure(0, weight=1)

        self.frames = {}

        # Crear una lista de clases de páginas
        pages = [HomePage, Page1, Page2]

        for page in pages:
            frame = page(container, self)
            self.frames[page] = frame
            frame.grid(row=0, column=0, sticky='nsew')

        self.show_frame(HomePage)

    def show_frame(self, page):
        frame = self.frames[page]
        frame.tkraise()
class HomePage(tk.Frame):

    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent)

        # Crear un botón en la página de inicio
           button1 = tk.Button(self, text='Ir a la página 1',
command=lambda: controller.show_frame(Page1))
        button1.pack(pady=10)
```

```
        # Crear un botón en la página de inicio
           button2 = tk.Button(self, text='Ir a la página 2',
command=lambda: controller.show_frame(Page2))
        button2.pack(pady=10)
class Page1(tk.Frame):

    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent)

        # Crear un botón en la página 1
          button = tk.Button(self, text='Volver a la página de
inicio', command=lambda: controller.show_frame(HomePage))
        button.pack(pady=10)
class Page2(tk.Frame):

    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent)

        # Crear un botón en la página 2
          button = tk.Button(self, text='Volver a la página de
inicio', command=lambda: controller.show_frame(HomePage))
        button.pack(pady=10)
if __name__ == '__main__':
    app = ExampleApp()
    app.mainloop()
```

La interfaz que generamos es la siguiente:

PyQT:

Principales características

1. Creación de ventanas y widgets: PyQt5 permite crear ventanas principales y secundarias en las que se puede diseñar la interfaz gráfica. Además de las ventanas, se pueden agregar una amplia variedad de widgets, como botones, etiquetas, campos de entrada, listas desplegables, tablas y muchos otros.
2. Diseño y posicionamiento de widgets: Los widgets en PyQt5 se pueden organizar utilizando técnicas de posicionamiento, como cuadrículas, cajas horizontales y verticales, y apilamiento. Esto facilita el diseño y la disposición de los elementos de la interfaz gráfica de manera flexible y ordenada.
3. Manejo de eventos: Permite asociar eventos, como clics de ratón, pulsaciones de teclas o cambios de valor, con funciones específicas. Esto permite controlar y responder a las acciones del usuario, como botones pulsados, selección de elementos de una lista, etc.
4. Estilos y apariencia personalizados: Proporciona una amplia gama de opciones de personalización de la apariencia de la interfaz gráfica. Se pueden aplicar estilos predefinidos, como los estilos de Windows, Fusión, macOS o estilos personalizados. También es posible personalizar colores, fuentes y otros aspectos visuales de los widgets.
5. Integración con Qt Designer: PyQt5 es compatible con Qt Designer, una herramienta visual para diseñar interfaces gráficas. Con Qt Designer, se pueden crear diseños de interfaz gráfica de manera visual y luego cargarlos en PyQt5 para agregar la lógica de programación y la funcionalidad.
6. Soporte para gráficos y multimedia: Ofrece funcionalidades avanzadas para la representación y visualización de gráficos y contenido multimedia. Proporciona widgets especiales para mostrar imágenes, reproducir vídeos y audio, así como también para crear gráficos interactivos utilizando la biblioteca QChart.
7. Integración con bases de datos: PyQt5 facilita la integración con bases de datos a través del módulo QtSQL. Proporciona clases y métodos para conectarse a bases de datos, ejecutar consultas SQL y manipular datos de manera eficiente.
8. Internacionalización: PyQt5 incluye funciones de internacionalización que permiten crear aplicaciones multilingües. Es posible cargar archivos de traducción para mostrar los textos de la interfaz gráfica en diferentes idiomas, lo que facilita la localización de la aplicación.

Principales clases y funciones

- ➔ QApplication: Esta clase representa la aplicación en sí misma. Se utiliza para inicializar la aplicación y manejar eventos de nivel superior.
- ➔ QWidget: Es la clase base para todos los componentes de la interfaz gráfica de usuario. Proporciona funcionalidades básicas como ventanas, botones, cuadros de texto, etiquetas, etc.

- ➔ QLayout: Esta clase se utiliza para organizar los widgets dentro de una ventana o un contenedor. Proporciona diferentes tipos de diseños, como QVBoxLayout, QHBoxLayout, QGridLayout, etc.
- ➔ QPushButton: Es un botón que puede ser presionado por el usuario para realizar una acción.
- ➔ QLabel: Se utiliza para mostrar texto o imágenes en una ventana.
- ➔ QLineEdit: Es un campo de texto en el que el usuario puede ingresar y editar texto.
- ➔ QCheckBox: Representa una casilla de verificación que puede ser seleccionada o deseleccionada.
- ➔ QRadioButton: Es un botón de opción que permite al usuario seleccionar una opción de un conjunto de opciones mutuamente excluyentes.
- ➔ QComboBox: Proporciona una lista desplegable de opciones donde el usuario puede seleccionar una.
- ➔ QSpinBox: Es un control que permite al usuario seleccionar un valor entero mediante un cuadro de texto y botones de aumento y disminución.
- ➔ QSlider: Representa un control deslizante que permite al usuario seleccionar un valor de un rango.
- ➔ QProgressBar: Muestra un indicador de progreso que puede ser utilizado para mostrar el avance de una tarea.
- ➔ QFileDialog: Proporciona un diálogo que permite al usuario seleccionar archivos o directorios.
- ➔ QMessageBox: Muestra un cuadro de diálogo con un mensaje y botones para aceptar o cancelar.

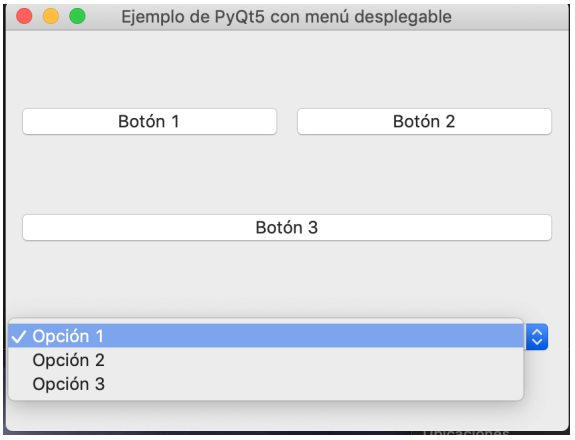
Ejemplo de código

Python

```python
import sys
from PyQt5.QtWidgets import QApplication, QMainWindow, QAction, QPushButton, QWidget, QGridLayout, QComboBox
class Example(QMainWindow):

    def __init__(self):
        super().__init__()

        # Crear un menú
        menu = self.menuBar().addMenu('Archivo')

        # Crear una acción para el menú
        exit_action = QAction('Salir', self)
        exit_action.triggered.connect(self.close)
        menu.addAction(exit_action)
```

```
        # Crear un widget central y una cuadrícula
        central_widget = QWidget(self)
        self.setCentralWidget(central_widget)
        grid_layout = QGridLayout(central_widget)

        # Crear botones
        button1 = QPushButton('Botón 1', self)
        button2 = QPushButton('Botón 2', self)
        button3 = QPushButton('Botón 3', self)

        # Agregar botones a la cuadrícula
        grid_layout.addWidget(button1, 0, 0)
        grid_layout.addWidget(button2, 0, 1)
        grid_layout.addWidget(button3, 1, 0, 1, 2)

        # Crear un menú desplegable
        combo_box = QComboBox(self)
        combo_box.addItem('Opción 1')
        combo_box.addItem('Opción 2')
        combo_box.addItem('Opción 3')

        # Agregar el menú desplegable a la cuadrícula
        grid_layout.addWidget(combo_box, 2, 0, 1, 2)

              self.setWindowTitle('Ejemplo de PyQt5 con menú
desplegable')
        self.show()
if __name__ == '__main__':
    app = QApplication(sys.argv)
    ex = Example()
    sys.exit(app.exec_())
```

La interfaz que generamos es la siguiente:



Paquetes en Java

Los paquetes de interfaces en Java, como Swing y AWT, son bibliotecas de programación que permiten a los desarrolladores crear interfaces gráficas de usuario (GUI) para sus aplicaciones. Estas bibliotecas proporcionan conjuntos de componentes visuales, como botones, etiquetas, cuadros de texto y paneles, que se utilizan para construir la interfaz gráfica de una aplicación.

La biblioteca AWT (Abstract Window Toolkit) fue la primera biblioteca de interfaces gráficas para Java. AWT se basa en componentes nativos del sistema operativo (SO) en el que se ejecuta la aplicación. Esto significa que los componentes AWT utilizan las API de interfaz gráfica proporcionadas por el SO subyacente, como Windows en sistemas operativos Windows o X Window System en sistemas operativos basados en Unix. AWT tiene la ventaja de una apariencia y comportamiento consistentes con las aplicaciones nativas del SO, pero puede tener limitaciones en términos de flexibilidad y apariencia.

Por otro lado, Swing es una biblioteca de interfaces gráficas más moderna y flexible que se basa completamente en Java. A diferencia de AWT, los componentes Swing son totalmente dibujados por software, lo que significa que tienen una apariencia y comportamiento consistentes en todos los sistemas operativos. Swing ofrece una amplia gama de componentes y permite una personalización más avanzada de la apariencia de la interfaz de usuario.

Swing:

Principales características

1. Componentes visuales: Swing ofrece una gran variedad de componentes visuales listos para usar, como botones, etiquetas, cuadros de texto, áreas de texto, casillas de verificación, botones de opción, listas desplegables, tablas y muchos más. Estos componentes se pueden combinar y personalizar para construir interfaces gráficas interactivas.
2. Contenedores: Swing proporciona contenedores, como paneles y marcos, que permiten organizar y colocar los componentes en la interfaz gráfica. Los contenedores pueden tener diferentes diseños, como diseño de cuadrícula (GridLayout), diseño de flujo (FlowLayout) o diseño de borde (BorderLayout), lo que facilita la estructuración de la interfaz.
3. Personalización de apariencia: Con Swing, puedes personalizar la apariencia de los componentes mediante el uso de plaf (pluggable look and feel). Puedes elegir entre diferentes estilos visuales, como el aspecto nativo del sistema operativo, aspectos temáticos o incluso crear tu propio aspecto personalizado.
4. Modelos y renderizado: Utiliza el patrón Modelo-Vista-Controlador (MVC), lo que significa que los componentes tienen un modelo subyacente que contiene los datos y la lógica de negocio, mientras que la vista se encarga de mostrar los datos y el

controlador gestiona las interacciones con los componentes. Además, muchos componentes permiten personalizar su renderizado para adaptarse a las necesidades específicas de la aplicación.

5. Eventos y acciones: Swing utiliza un modelo de eventos para manejar la interacción del usuario con los componentes. Puedes registrar escuchadores de eventos para responder a acciones específicas, como hacer clic en un botón, escribir en un campo de texto o seleccionar un elemento de una lista. Esto permite que las aplicaciones respondan de forma dinámica a las interacciones del usuario.
6. Hilos y SwingWorker: Dado que las interfaces gráficas son interactivas y deben responder rápidamente, Swing ofrece mecanismos para realizar operaciones intensivas en la CPU o que puedan bloquear la interfaz en hilos separados. El SwingWorker es una clase que facilita la ejecución de tareas en segundo plano mientras se mantiene la capacidad de actualizar la interfaz gráfica de forma segura.

Principales clases y funciones

- ➔ JFrame: Es una clase que se utiliza para crear una ventana que puede contener otros componentes, como botones, cuadros de texto, etc.
- ➔ JPanel: Es una clase que se utiliza para crear un contenedor que se puede utilizar para organizar y colocar componentes.
- ➔ JButton: Es una clase que se utiliza para crear botones que pueden ser utilizados para activar funciones de la aplicación cuando se hace clic en ellos.
- ➔ JCheckBox: Es una clase que se utiliza para crear casillas de verificación que permiten al usuario seleccionar una o varias opciones.
- ➔ JTextField: Es una clase que se utiliza para crear cuadros de texto que permiten al usuario ingresar y editar texto.
- ➔ JTable: Es una clase que se utiliza para crear una tabla que puede mostrar datos en una forma estructurada y fácil de leer.
- ➔ JMenuBar: Es una clase que se utiliza para crear un menú que se puede agregar a un JFrame o un JDialog.
- ➔ JTree: Es una clase que se utiliza para crear un árbol de nodos que puede ser utilizado para organizar y presentar información jerárquica.

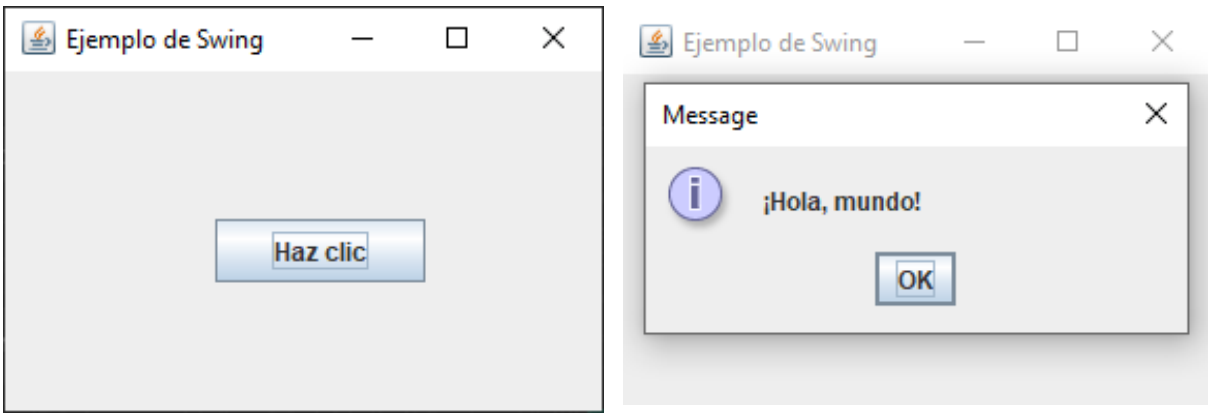
Ejemplo de código

Java

```java
import javax.swing.*;

public class SwingExample {
    public static void main(String[] args) {
        // Crear una ventana JFrame
        JFrame frame = new JFrame("Ejemplo de Swing");
```

```
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(300, 200);

        // Crear un botón JButton
        JButton button = new JButton("Haz clic");
        button.setBounds(100, 70, 100, 30);

        // Agregar un ActionListener al botón
        button.addActionListener(e -> {
                  JOptionPane.showMessageDialog(frame, "¡Hola,
mundo!");
        });

        // Agregar el botón a la ventana
        frame.add(button);

        // Mostrar la ventana
        frame.setLayout(null);
        frame.setVisible(true);
    }
}
```

La interfaz que generamos es la siguiente:



AWT:

Principales características

1. Componentes visuales: AWT proporciona una variedad de componentes visuales, como botones, etiquetas, campos de texto, áreas de texto, casillas de verificación, botones de opción, listas desplegables y paneles. Estos componentes se utilizan para construir la interfaz gráfica de usuario de una aplicación.
2. Contenedores: AWT ofrece contenedores, como ventanas y paneles, que permiten organizar y colocar los componentes en la interfaz gráfica. Los contenedores pueden tener diferentes diseños, como diseño de cuadrícula (GridLayout) o diseño de borde (BorderLayout), lo que facilita la estructuración de la interfaz.
3. Apariencia nativa: Utiliza componentes nativos del sistema operativo en el que se ejecuta la aplicación. Esto significa que los componentes AWT se ven y se comportan de acuerdo con las pautas de diseño de la interfaz gráfica del SO subyacente. Esto proporciona una apariencia consistente con las aplicaciones nativas del sistema operativo.
4. Gráficos: También permite la creación y manipulación de gráficos en la interfaz gráfica. Puedes dibujar formas geométricas, imágenes y texto en componentes AWT utilizando objetos Graphics y Graphics2D. Esto permite crear elementos visuales personalizados y realizar operaciones de dibujo avanzadas.
5. Eventos y acciones: Utiliza un modelo de eventos para manejar la interacción del usuario con los componentes. Puedes registrar escuchadores de eventos para responder a acciones específicas, como hacer clic en un botón, escribir en un campo de texto o seleccionar un elemento de una lista. Esto permite que las aplicaciones respondan de forma dinámica a las interacciones del usuario.
6. Hilos y Event Dispatch Thread (EDT): Para manejar los eventos y las actualizaciones de la interfaz gráfica, hace uso del Event Dispatch Thread (EDT). Es importante realizar todas las modificaciones de la interfaz gráfica en el EDT para garantizar un funcionamiento correcto y evitar problemas de concurrencia.
7. Integración con el sistema operativo: AWT se basa en las API de interfaz gráfica proporcionadas por el sistema operativo subyacente. Esto permite una estrecha integración con el SO y el aprovechamiento de las funcionalidades específicas del mismo, como las fuentes, los colores y los elementos de la interfaz nativa.

Principales clases y funciones

- ➔ Frame: Es una clase que representa una ventana en la interfaz de usuario. Se utiliza como contenedor para otros componentes.
- ➔ Panel: Es una clase que se utiliza como contenedor para organizar y colocar componentes.
- ➔ Button: Es una clase que se utiliza para crear botones en la interfaz de usuario.
- ➔ Checkbox: Es una clase que se utiliza para crear casillas de verificación que permiten al usuario seleccionar opciones.

- ➔ TextField: Es una clase que se utiliza para crear cuadros de texto en los que el usuario puede ingresar y editar texto.
- ➔ Label: Es una clase que se utiliza para mostrar texto o imágenes en la interfaz de usuario.
- ➔ MenuBar: Es una clase que se utiliza para crear una barra de menú que contiene opciones de menú.
- ➔ List: Es una clase que se utiliza para crear una lista de elementos seleccionables.
- ➔ Choice: Es una clase que se utiliza para crear una lista desplegable de opciones.
- ➔ Canvas: Es una clase que se utiliza como un lienzo en el que se pueden dibujar gráficos personalizados.

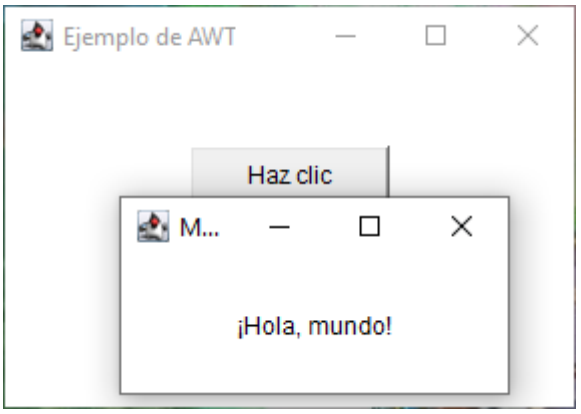
Ejemplo de código

Java

```java
import java.awt.*;
import java.awt.event.*;

public class AWTExample {
    public static void main(String[] args) {
        // Crear un Frame
        Frame frame = new Frame("Ejemplo de AWT");
        frame.setSize(300, 200);

        // Crear un Button
        Button button = new Button("Haz clic");
        button.setBounds(100, 70, 100, 30);

        // Agregar un ActionListener al botón
        button.addActionListener(new ActionListener() {
           public void actionPerformed(ActionEvent e) {
                Frame messageFrame = new Frame("Mensaje");
                messageFrame.setSize(200,100);
                messageFrame.setLayout(new BorderLayout());

                Label messageLabel = new Label("¡Hola,
          mundo!");
                messageLabel.setAlignment(Label.CENTER);
                messageFrame.add(messageLabel,
          BorderLayout.CENTER);
                messageFrame.setVisible(true);
```

```
            }
        });

        // Agregar el botón al Frame
        frame.add(button);

        // Mostrar el Frame
        frame.setLayout(null);
        frame.setVisible(true);
    }
}
```

La interfaz que generamos es la siguiente:



Conclusión

Tras investigar y explorar las librerías de interfaces de diferentes lenguajes de programación. Estas librerías desempeñan un papel fundamental en el desarrollo de aplicaciones con interfaces gráficas, y cada una de ellas tiene sus propias características y ventajas distintivas.

Cada una de estas librerías tiene sus fortalezas y debilidades particulares, y la elección entre ellas dependerá de los requisitos del proyecto y las preferencias del desarrollador. Al seleccionar una librería de interfaz, es importante considerar factores como la facilidad de uso, la flexibilidad, la compatibilidad con plataformas, el soporte de la comunidad y las necesidades específicas del proyecto.

Es decir, las librerías Tkinter, PyQt5, Swing y AWT ofrecen a los desarrolladores una amplia gama de opciones para crear interfaces gráficas en Python y Java. Estas herramientas proporcionan una base sólida para diseñar aplicaciones intuitivas y atractivas, permitiendo a los programadores enfocarse en la funcionalidad y la experiencia del usuario. La elección de la librería adecuada dependerá de las necesidades y preferencias individuales, pero todas ellas son poderosas herramientas que contribuyen al desarrollo de software de calidad con interfaces gráficas impresionantes.