PYQT

Qué tenemos que aprender:

1. Genera interfaces gráficos de usuario mediante editores visuales utilizando las funcionalidades del editor y adaptando el código generado.

- a) Se han analizado las herramientas y librerías disponibles para la generación de interfaces gráficos.
- b) Se ha creado un interfaz gráfico utilizando las herramientas de un editor visual.
- c) Se han utilizado las funciones del editor para ubicar los componentes del interfaz.
- d) Se han modificado las propiedades de los componentes para adecuarlas a las necesidades de la aplicación.
- e) Se ha analizado el código generado por el editor visual.
- f) Se ha modificado el código generado por el editor visual.
- g) Se han asociado a los eventos las acciones correspondientes.
- h) Se ha desarrollado una aplicación que incluye el interfaz gráfico obtenido

1. APLICACIÓN BASE

```
import sys
from PyQt6.QtWidgets import QApplication, QMainWindow, QWidget
from PyQt6.QtGui import QPalette, QColor

class MainWindow(QMainWindow):

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

        self.setWindowTitle("My App")

app = QApplication(sys.argv)
window = MainWindow()
window.show()

app.exec()
```

2.WIDGETS

```
import sys

from PyQt6.QtCore import Qt
from PyQt6.QtWidgets import (
    QApplication,
    QCheckBox,
    QComboBox,
    QDateEdit,
    QDateTimeEdit,
    QDial,
    QDoubleSpinBox,
    QFontComboBox,
    QLabel,
    QLCDNumber,
    QLineEdit,
    QMainWindow,
    QProgressBar,
    QPushButton,
    QRadioButton,
    QSlider,
    QSpinBox,
    QTimeEdit,
    QVBoxLayout,
    QWidget,
)

# Subclass QMainWindow to customize your application's main window
class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()

        self.setWindowTitle("Widgets App")

        layout = QVBoxLayout()
        widgets = [
            QCheckBox,
            QComboBox,
            QDateEdit,
            QDateTimeEdit,
            QDial,
            QDoubleSpinBox,
            QFontComboBox,
            QLCDNumber,
            QLabel,
            QLineEdit,
            QProgressBar,
            QPushButton,
            QRadioButton,
            QSlider,
            QSpinBox,
            QTimeEdit,
        ]

        for w in widgets:
            layout.addWidget(w())

        widget = QWidget()
        widget.setLayout(layout)

        # Set the central widget of the Window. Widget will expand
        # to take up all the space in the window by default.
        self.setCentralWidget(widget)

app = QApplication(sys.argv)
window = MainWindow()
window.show()
app.exec()
```

crea un label para poner una etiqueta con alineamiento horizontal y vertical

Alineado horizontal

- Qt.AlignmentFlag.AlignLeft Alineado a la izquierda del borde.
- Qt.AlignmentFlag.AlignRight Alineado a la derecha del borde.
- Qt.AlignmentFlag.AlignHCenter Centrado horizaontalmente en el espacio disponible.
- Qt.AlignmentFlag.AlignJustify Justifica el texto en el espacio que hay disponible.

Alineado vertical

- Qt.AlignmentFlag.AlignTop Alineado
- Qt.AlignmentFlag.AlignBottom Alineado con el fondo.
- Qt.AlignmentFlag.AlignVCenter Centrado verticalmente en el espacio disponible.

  Cuando queremos combinar diferentes tipos de alineamiento tenemos que utilizar el OR

solución

```
import sys

from PyQt6.QtCore import Qt
from PyQt6.QtWidgets import (
    QApplication,
    QCheckBox,
    QComboBox,
    QDateEdit,
    QDateTimeEdit,
    QDial,
    QDoubleSpinBox,
    QFontComboBox,
    QLabel,
    QLCDNumber,
    QLineEdit,
    QMainWindow,
    QProgressBar,
    QPushButton,
    QRadioButton,
    QSlider,
    QSpinBox,
    QTimeEdit,
    QVBoxLayout,
    QWidget,
)

# Subclass QMainWindow to customize your application's main window
class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()

        self.setWindowTitle("Widgets App")

        layout = QVBoxLayout()
        widgets = [
            QCheckBox,
            QComboBox,
            QDateEdit,
            QDateTimeEdit,
            QDial,
            QDoubleSpinBox,
            QFontComboBox,
            QLCDNumber,
            QLabel,
            QLineEdit,
            QProgressBar,
            QPushButton,
            QRadioButton,
            QSlider,
            QSpinBox,
            QTimeEdit,
        ]

        #for w in widgets:
        #    layout.addWidget(w())

        label = QLabel()
        label.setText ("hola")
        layout.addWidget(label)
        label.setAlignment(Qt.AlignmentFlag.AlignHCenter | Qt.AlignmentFlag.AlignVCenter)

        label2 = QLabel()
        label2.setText ("hola")
        layout.addWidget(label2)
        label2.setAlignment(Qt.AlignmentFlag.AlignHCenter | Qt.AlignmentFlag.AlignVCenter)

        widget = QWidget()
        widget.setLayout(layout)

        # Set the central widget of the Window. Widget will expand
        # to take up all the space in the window by default.
        self.setCentralWidget(widget)

app = QApplication(sys.argv)
window = MainWindow()
window.show()
app.exec()
```

```
        widget = QLabel("Hello")

        font = widget.font()
        font.setPointSize(30)
        widget.setFont(font)
        widget.setAlignment(Qt.AlignmentFlag.AlignHCenter | Qt.AlignmentFlag.AlignVCenter)

        self.setCentralWidget(widget)
```

Ejercicio 1. Prueba otras orientaciones usando el or

Ejercicio 2. Prueba a añadir más labels en horizontal y en vertical.

También podemos añadir imágenes a los QLabel

```
label = QLabel(self)
pixmap = QPixmap('image.jpeg')
label.setPixmap(pixmap)

# Optional, resize window to image size
self.resize(pixmap.width(),pixmap.height())
```

Propiedades del QLabel:
https://doc.qt.io/qtforpython-6/PySide6/QtWidgets/QLabel.html

QCheckBox

Es un checkbox, vitaminado puedes hacer un tricheck con estado parcial de chequeado

```
        widget = QCheckBox()
        widget.setCheckState(Qt.CheckState.Checked)

        # For tristate: widget.setCheckState(Qt.CheckState.PartiallyChecked)
        # Or: widget.setTristate(True)
        widget.stateChanged.connect(self.show_state)

        self.setCentralWidget(widget)

    def show_state(self, s):
        print(s == Qt.CheckState.Checked.value)
```

Ejercicio para conectar con otro estado para ver si está chequeado o no

```

# señal para detectar cambios en la casilla
casilla.stateChanged.connect(self.estado_cambiado)

def estado_cambiado(self, estado):
    print(estado)
```

Ejercicio verifica cuando está chequeado y cuando no para printear su estado

QComboBox

Es una lista desplegable, vamos a probarla

```
# creamos un desplegable
        desplegable = QComboBox()
        self.setCentralWidget(desplegable)

        desplegable.addItems(["Opción 1", "Opción 2", "Opción 3"])
```

crea dos métodos para obtener la información del índice y el elemento seleccionado en la lista desplegable

Ejercicio: A partir de las señales del método. crea dos métodos para obtener la información del índice y el elemento seleccionado en la lista desplegable

widget.currentIndexChanged.connect(self.index_changed)
widget.currentTextChanged.connect(self.text_changed)

QLineEdit

Prueba este ejemplo, y modificalo para crear una pantalla de login que sea operativa

```
from PyQt5.QtWidgets import QApplication,QLineEdit,QWidget,QFormLayout
from PyQt5.QtGui import QIntValidator,QDoubleValidator,QFont
from PyQt5.QtCore import Qt
import sys

class lineEditDemo(QWidget):
        def __init__(self,parent=None):
                super().__init__(parent)
                e1 = QLineEdit()
                e1.setValidator(QIntValidator())
                e1.setMaxLength(4)
                e1.setAlignment(Qt.AlignRight)
                e1.setFont(QFont("Arial",20))

                e2 = QLineEdit()
                e2.setValidator(QDoubleValidator(0.99,99.99,2))
                e3 = QLineEdit()
                e3.setInputMask("+99_9999_999999")

                e4 = QLineEdit()
                e4.textChanged.connect(self.textchanged)

                e5 = QLineEdit()
                e5.setEchoMode(QLineEdit.Password)

                e6 = QLineEdit("Hello PyQt5")
                e6.setReadOnly(True)
                e5.editingFinished.connect(self.enterPress)

                flo = QFormLayout()
                flo.addRow("integer validator",e1)
                flo.addRow("Double validator",e2)
                flo.addRow("Input Mask",e3)
                flo.addRow("Text changed",e4)
                flo.addRow("Password",e5)
                flo.addRow("Read Only",e6)

                self.setLayout(flo)
                self.setWindowTitle("QLineEdit Example")

        def textchanged(self,text):
                print("Changed: " + text)

        def enterPress(self):
                print("Enter pressed")

if __name__ == "__main__":
        app = QApplication(sys.argv)
        win = lineEditDemo()
        win.show()
        sys.exit(app.exec_())
```

Los botones son QPushButton
Para añadir los botones que controlen login y cancel puedes ver el siguiente enlace:
https://pythonpyqt.com/pyqt-button/

3. TIPOS DE LAYOUT

LAYOUTS, DIALOGS Y MENÚS

APLICACIÓN BASE

```
import sys
from PyQt6.QtWidgets import QApplication, QMainWindow, QWidget
from PyQt6.QtGui import QPalette, QColor

class MainWindow(QMainWindow):

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

        self.setWindowTitle("My App")

app = QApplication(sys.argv)
window = MainWindow()
window.show()

app.exec()
```

Ejemplo Layout 1

```
class Color(QWidget):

    def __init__(self, color):
        super(Color, self).__init__()
        self.setAutoFillBackground(True)

        palette = self.palette()
        palette.setColor(QPalette.ColorRole.Window, QColor(color))
        self.setPalette(palette)

class MainWindow(QMainWindow):

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

        self.setWindowTitle("My App")

        layout = QVBoxLayout()

        layout.addWidget(Color('red'))
        layout.addWidget(Color('green'))
        layout.addWidget(Color('blue'))

        widget = QWidget()
        widget.setLayout(layout)
        self.setCentralWidget(widget)
```

Ejercicio 1. Añade componentes diferentes a los widget y muestrame la pantalla

Ejemplo Layout 2

```
class MainWindow(QMainWindow):
    class Color(QWidget):

    def __init__(self, color):
        super(Color, self).__init__()
        self.setAutoFillBackground(True)

        palette = self.palette()
        palette.setColor(QPalette.ColorRole.Window, QColor(color))
        self.setPalette(palette)

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

        self.setWindowTitle("My App")

        layout = QHBoxLayout()

        layout.addWidget(Color('red'))
        layout.addWidget(Color('green'))
        layout.addWidget(Color('blue'))

        widget = QWidget()
        widget.setLayout(layout)
        self.setCentralWidget(widget)
```

Ejercicio 2. pon un Qlabel y un QLineEdit

Ejemplo de Layout 3 mezcla de Horizontal y Vertical

```
class Color(QWidget):

    def __init__(self, color):
        super(Color, self).__init__()
        self.setAutoFillBackground(True)

        palette = self.palette()
        palette.setColor(QPalette.ColorRole.Window, QColor(color))
        self.setPalette(palette)

class MainWindow(QMainWindow):

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

        self.setWindowTitle("My App")

        layout1 = QHBoxLayout()
        layout2 = QVBoxLayout()
        layout3 = QVBoxLayout()

        layout1.setContentsMargins(0,0,0,0)
        layout1.setSpacing(20)

        layout2.addWidget(Color('red'))
        layout2.addWidget(Color('yellow'))
        layout2.addWidget(Color('purple'))

        layout1.addLayout( layout2 )

        layout1.addWidget(Color('green'))

        layout3.addWidget(Color('red'))
        layout3.addWidget(Color('purple'))

        layout1.addLayout( layout3 )

        widget = QWidget()
        widget.setLayout(layout1)
        self.setCentralWidget(widget)
```

Ejercicio modifica el nested layout a otra configuración con los colores ;)

Ejemplo de GridLayout en formato tabla

```
class Color(QWidget):

    def __init__(self, color):
        super(Color, self).__init__()
        self.setAutoFillBackground(True)

        palette = self.palette()
        palette.setColor(QPalette.ColorRole.Window, QColor(color))
        self.setPalette(palette)

class MainWindow(QMainWindow):

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

        self.setWindowTitle("My App")

        layout = QGridLayout()

        layout.addWidget(Color('red'), 0, 0)
        layout.addWidget(Color('green'), 1, 0)
        layout.addWidget(Color('blue'), 1, 1)
        layout.addWidget(Color('purple'), 2, 1)

        widget = QWidget()
        widget.setLayout(layout)
        self.setCentralWidget(widget)
```

Ejercicio Añade widgets a esta configuración y luego cambia la configuración con otra estructura.

Ejercicio Crea un Vertical Layout y un Horizontal Layout Añade 3 botones QButtons en la primera línea y en la otra crea un widget de color

Ejemplo de StackedLayout

```
from PyQt6.QtWidgets import QStackedLayout  # add this import

class Color(QWidget):

    def __init__(self, color):
        super(Color, self).__init__()
        self.setAutoFillBackground(True)

        palette = self.palette()
        palette.setColor(QPalette.ColorRole.Window, QColor(color))
        self.setPalette(palette)

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()

        self.setWindowTitle("My App")

        layout = QStackedLayout()

        layout.addWidget(Color("red"))
        layout.addWidget(Color("green"))
        layout.addWidget(Color("blue"))
        layout.addWidget(Color("yellow"))

        layout.setCurrentIndex(3)

        widget = QWidget()
        widget.setLayout(layout)
        self.setCentralWidget(widget)
```

Experiencia - Cambia el setCurrentIndex a 1 - ¿Qué ocurre cómo funciona este layout?

Ejercicio en el ejercio anterior de los tres botones añade una fila de abajo un stackedlayout como en el ejemplo anteior. Los botónes les pones los nombres de los colores y tienes que asociar la señal de click "nombre_del_botón.pressed.connect" para que cuando pulses cambies el setCurrentindex de stacklayout y que aparezca el color que quieres que salga.

Clase del día 1/10/2024

```
import sys
from PyQt6.QtWidgets import (QApplication, QWidget,
                             QPushButton,QLineEdit,QGridLayout)

class calculadora(QWidget):

    def metodo1():
        pass

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

        self.setWindowTitle("LA calcula de la liga Fantasia")
        self.setGeometry (600,100,300,300)

        self.resultado = QLineEdit()
        layout = QGridLayout()
        layout.addWidget(self.resultado,0,0,1,5)
        nombre_boton = [["1","2","3","+"],
                        ["4","5","6","-"],
                        ["7","8","9", "*"],
                        ["0","/",".","="]]
        # añadir componentes.
        for i in range (1,5):
            for j in range (4):
                boton = QPushButton(nombre_boton[i-1][j])
                layout.addWidget(boton,i,j)
                boton.clicked.connect(self.press_button)

        self.setLayout(layout)

    def press_button (self):
        sender = self.sender()
        if (sender.text() == "=" ):
            try:

                self.resultado.setText(str(eval(self.resultado.text())))
            except (Exception):
                self.resultado.setText("operación inválida")
        else:
            self.resultado.setText(self.resultado.text() + sender.text())


if __name__ == "__main__":

    app = QApplication(sys.argv)
    calc = calculadora()
    calc.show()
    sys.exit(app.exec())
```

clase 04/10/2024

Ejercicio: Crea un formulario de login

```
import sys
from PyQt6.QtWidgets import (QApplication, QLineEdit,
                             QWidget, QPushButton, QGridLayout, QLabel)
from PyQt6.QtGui import QIcon


class login (QWidget):

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

        # Configuración inicial de la ventana
        self.setWindowTitle("login ejemplo")
        self.setGeometry (600,100,300,300)
        # self.setWindowIcon (UIcon ("ruta del fichero .ico"))

        # creamos el grid
        layout = QGridLayout()

        # creamos los componentes visuales
        nombrelabel = QLabel ("login")
        nombre = QLineEdit ()

        # Ejemplo de como Añadir un place holder
        nombre.setPlaceholderText ("pon aqui tu login")
        passwordlabel = QLabel ("password")
        password = QLineEdit()

        # Añadimos los componentes al layout
        layout.addWidget(nombrelabel,0,0)
        layout.addWidget (nombre, 0,1)
        layout.addWidget(passwordlabel,1,0)
        layout.addWidget (password,1,1,1,1)
        btnLogin = QPushButton ("login")
        btnCancel = QPushButton ("cancel")
        layout.addWidget (btnLogin, 2,0)
        layout.addWidget (btnCancel, 2,1)

        # Añadimos el layout a QWidget
        self.setLayout (layout)

if __name__ == "__main__":
    app = QApplication(sys.argv)
    dialogo_login = login()
    dialogo_login.show()
    sys.exit(app.exec())
```

Cómo poner un text con password

Ejemplo:

```
pw = QtGui.QLineEdit()
pw.setEchoMode(QtGui.QLineEdit.Password)
```

Diálogos y otras ventanas

Esqueleto

Probad primero esta aplicación.

```
import sys

from PyQt6.QtWidgets import QApplication, QMainWindow, QPushButton


class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()

        self.setWindowTitle("My App")

        button = QPushButton("Press me for a dialog!")
        button.clicked.connect(self.button_clicked)
        self.setCentralWidget(button)

    def button_clicked(self, s):
        print("click", s)


app = QApplication(sys.argv)
window = MainWindow()
window.show()
app.exec()
```

Al pulsar el botón nos creamos un diálogo modal

```
import sys

from PyQt6.QtWidgets import QApplication, QDialog, QMainWindow, QPushButton


class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()

        self.setWindowTitle("My App")

        button = QPushButton("Press me for a dialog!")
        button.clicked.connect(self.button_clicked)
        self.setCentralWidget(button)

    def button_clicked(self, s):
        print("click", s)

        dlg = QDialog(self)
        dlg.setWindowTitle("HELLO!")
        dlg.exec()


app = QApplication(sys.argv)
window = MainWindow()
window.show()
app.exec()
```

Una vez probado estos ejemplos como ejercicio vamos a añadir nuestro proprio diálogo.

Creación de un diálogo personalizado por nosotros

Que muestra un label en el diálogo

```
class CustomDialog(QDialog):
    def __init__(self):
        super().__init__()

        self.setWindowTitle("HELLO!")

        layout = QVBoxLayout()
        message = QLabel("Something happened, is that OK?")
        layout.addWidget(message)
        layout.addWidget(self.buttonBox)
        self.setLayout(layout)
```

Ejercicio:

Hay que añadir el diálogo personalizado a la aplicación anterior que al pulsar el botón nos muestra este diálogo

Solución:

```
import sys

from PyQt6.QtWidgets import QApplication, QDialog, QMainWindow, QPushButton, QVBoxLayout, QLabel

class CustomDialog(QDialog):
    def __init__(self):
        super().__init__()

        self.setWindowTitle("HELLO!")

        layout = QVBoxLayout()
        message = QLabel("Something happened, is that OK?")
        message.setObjectName('nom_plan_label')
        message.setStyleSheet('QLabel#nom_plan_label {color: red}')
        #message.setStyleSheet('QLabel#nom_plan_label {color: blue}')
        layout.addWidget(message)
        self.setLayout(layout)

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()

        self.setWindowTitle("My App")

        button = QPushButton("Press me for a dialog!")
        button.setCheckable (True)
        button.clicked.connect(self.button_clicked)
        self.setCentralWidget(button)

    def button_clicked(self, s):

        dlg = CustomDialog()
        dlg.exec()


app = QApplication(sys.argv)
window = MainWindow()
window.show()
app.exec()
```

Modificación añadiendo setCheckeable para cambiar el color de Qlabel de nuestro diálogo personalizado

Cada vez que pulsamos al botón nos cambia el color del Qlabel del CustomDialog

```
import sys

from PyQt6.QtWidgets import QApplication, QDialog, QMainWindow, QPushButton, QVBoxLayout, QLabel

class CustomDialog(QDialog):
    def __init__(self,isColor):
        super().__init__()

        self.setWindowTitle("HELLO!")

        layout = QVBoxLayout()
        message = QLabel("Something happened, is that OK?")
        message.setObjectName('nom_plan_label')
        if (isColor):
            message.setStyleSheet('QLabel#nom_plan_label {color: red}')
        else:
            message.setStyleSheet('QLabel#nom_plan_label {color: blue}')
        layout.addWidget(message)
        self.setLayout(layout)

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()

        self.setWindowTitle("My App")

        button = QPushButton("Press me for a dialog!")
        button.setCheckable (True)
        button.clicked.connect(self.button_clicked)
        self.setCentralWidget(button)

    def button_clicked(self, s):

        dlg = CustomDialog(s)
        dlg.exec()


app = QApplication(sys.argv)
window = MainWindow()
window.show()
app.exec()
```

Añadir botones al CustomDialog

```
import sys

from PyQt6.QtWidgets import QApplication, QDialog, QMainWindow, QPushButton, QVBoxLayout, QLabel, QDialogButtonBox

class CustomDialog(QDialog):
    def __init__(self,isColor):
        super().__init__()

        self.setWindowTitle("HELLO!")

        QBtn = (
            QDialogButtonBox.StandardButton.Ok | QDialogButtonBox.StandardButton.Cancel
        )

        self.buttonBox = QDialogButtonBox(QBtn)
        self.buttonBox.accepted.connect(self.accept)
        self.buttonBox.rejected.connect(self.reject)

        layout = QVBoxLayout()
        message = QLabel("Something happened, is that OK?")
        message.setObjectName('nom_plan_label')
        if (isColor):
            message.setStyleSheet('QLabel#nom_plan_label {color: red}')
        else:
            message.setStyleSheet('QLabel#nom_plan_label {color: blue}')
        layout.addWidget(message)
        layout.addWidget (self.buttonBox)
        self.setLayout(layout)

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()

        self.setWindowTitle("My App")

        button = QPushButton("Press me for a dialog!")
        button.setCheckable (True)
        button.clicked.connect(self.button_clicked)
        self.setCentralWidget(button)

    def button_clicked(self, s):

        dlg = CustomDialog(s)
        dlg.exec()


app = QApplication(sys.argv)
window = MainWindow()
window.show()
app.exec()
```

tipos de botones

- QMessageBox.StandardButton.Ok
- QMessageBox.StandardButton.Open
- QMessageBox.StandardButton.Save
- QMessageBox.StandardButton.Cancel
- QMessageBox.StandardButton.Close
- QMessageBox.StandardButton.Discard
- QMessageBox.StandardButton.Apply
- QMessageBox.StandardButton.Reset
- QMessageBox.StandardButton.RestoreDefaults
- QMessageBox.StandardButton.Help
- QMessageBox.StandardButton.SaveAll
- QMessageBox.StandardButton.Yes
- QMessageBox.StandardButton.YesToAll
- QMessageBox.StandardButton.No
- QMessageBox.StandardButton.NoToAll
- QMessageBox.StandardButton.Abort
- QMessageBox.StandardButton.Retry
- QMessageBox.StandardButton.Ignore
- QMessageBox.StandardButton.NoButton

Ejemplo rayada de código con QDialog y QMessageBox

```
import sys

from PyQt6.QtWidgets import (QApplication, QDialog, QMainWindow, QPushButton,
                             QVBoxLayout, QLabel, QDialogButtonBox, QMessageBox)

class CustomDialog(QDialog):
    def __init__(self,isColor):
        super().__init__()

        self.setWindowTitle("HELLO!")

        QBtn = (
            QDialogButtonBox.StandardButton.Yes | QDialogButtonBox.StandardButton.No
        )

        self.buttonBox = QDialogButtonBox(QBtn)
        self.buttonBox.accepted.connect(self.accept)
        self.buttonBox.rejected.connect(self.mensajealerta)

        layout = QVBoxLayout()
        message = QLabel("Something happened, is that OK?")
        message.setObjectName('nom_plan_label')
        if (isColor):
            message.setStyleSheet('QLabel#nom_plan_label {color: red}')
        else:
            message.setStyleSheet('QLabel#nom_plan_label {color: blue}')
        layout.addWidget(message)
        layout.addWidget (self.buttonBox)
        self.setLayout(layout)

    def mensajealerta (self):
      QMessageBox.warning(self, "title", "mensaje")


class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()

        self.setWindowTitle("My App")

        button = QPushButton("Press me for a dialog!")
        button.setCheckable (True)
        button.clicked.connect(self.button_clicked)
        self.setCentralWidget(button)

    def button_clicked(self, s):
        dlg = CustomDialog(s)
        dlg.exec()



app = QApplication(sys.argv)
window = MainWindow()
window.show()
app.exec()
```

QtDesigner

https://build-system.fman.io/qt-designer-download

creación de ventanas con pyqt y qt designer

Crea diálogos y ventanas con qt designer

1. Crea una ventana con qt designer. Añadimos un botón qPushbutton y un qlabel

2. guardar como ejemploqt.ui dentro de la carpeta donde vamos a crear el código

3. Vamos a crear el código python (copiar el código que te pongo a continuación

```
#importamos las librerias necesarias
import sys
from PyQt6 import QtWidgets, uic

#Carga la interfaz gráfica y conecta los botones
class Ventana(QtWidgets.QMainWindow):
    '''Esta es la clase principal'''
    #Inicializamos la ventana y conectamos los botones
    def __init__(self, padre=None):
        #Inicializa la ventana
        QtWidgets.QMainWindow.__init__(self, padre)
        uic.loadUi("Ejemploqt.ui",self) #Lee el archivo de QtDesigner

        self.setWindowTitle("Ejemplo") #Titulo de la ventana

        #Conectar botón a función
        self.pushButton.clicked.connect(self.funcion)

    def funcion(self):
        if self.label.text() == "":
            self.label.setText("Hola clase")
        else:
            self.label.setText("")


# se crea la instancia de la aplicación
app = QtWidgets.QApplication(sys.argv)
# se crea la instancia de la ventana
miVentana = Ventana()
# se muestra la ventana
miVentana.show()
# se entrega el control al sistema operativo
sys.exit(app.exec())
```

Conexión de los diseños ui con la parte de código de python

¿Dónde se une la parte gráfica ui con el código?

```
        QtWidgets.QMainWindow.__init__(self, padre)
        uic.loadUi("Ejemploqt.ui",self) #Lee el archivo de QtDesigner
```

Control del botón gráfico con las acciones en código

Conectar el botón con la función

```
self.pushButton.clicked.connect(self.funcion)
```

Función para realizar una acción cuando realizan click

```
    def funcion(self):
        if self.label.text() == "":
            self.label.setText("Hola clase")
        else:
            self.label.setText("")
```

Ejemplo de barra de progreso

```
#importamos las librerias necesarias
import sys, time
from PyQt6 import QtWidgets, uic

#Carga la interfaz gráfica y conecta los botones
class Ventana(QtWidgets.QMainWindow):
    '''Esta es la clase principal'''
    #Inicializamos la ventana y conectamos los botones
    def __init__(self, padre=None):
        #Inicializa la ventana
        QtWidgets.QMainWindow.__init__(self, padre)
        uic.loadUi("barraprogreso.ui",self) #Lee el archivo de QtDesigner

        self.setWindowTitle("Ejemplo") #Titulo de la ventana

        #setear la barra de progreso
        self.progressBar

        #Conectar botón a función
        self.pushButton.clicked.connect(self.automatico)
        self.current_value = 0

    def funcion(self):
        if self.current_value <= self.progressBar.maximum():
            self.current_value += 5
            self.progressBar.setValue(self.current_value)

    def automatico (self):
        for i in range (20):
            time.sleep(1)
            self.current_value += 5
            self.progressBar.setValue(self.current_value)



# se crea la instancia de la aplicación
app = QtWidgets.QApplication(sys.argv)
# se crea la instancia de la ventana
miVentana = Ventana()
# se muestra la ventana
miVentana.show()
# se entrega el control al sistema operativo
sys.exit(app.exec())
```

Ejercicio 1

Crea una pantalla con dos botones en el primer botón queremos que modifique algún otro elemento visual y el segundo otro.

Solución

```

```

Ejercicio 2: Crea dos ventanas con qt designer que registre información y que con un botón podamos mostrar la informacióno de otra

Ejemplo: Cómo mostrar de una ventana otra.

Hay dos clases cada clase representa cada ventana y desde la ventana principal hay un botón que llama a la nueva ventana.
Modifica el siguiente código para conseguir mostrar tus ventanas diseñadas con qt designer.

```
from PyQt6.QtWidgets import QApplication, QMainWindow, QPushButton, QLabel, QVBoxLayout, QWidget

import sys


class AnotherWindow(QWidget):
    """
    This "window" is a QWidget. If it has no parent, it
    will appear as a free-floating window as we want.
    """
    def __init__(self):
        super().__init__()
        layout = QVBoxLayout()
        self.label = QLabel("Another Window")
        layout.addWidget(self.label)
        self.setLayout(layout)


class MainWindow(QMainWindow):

    def __init__(self):
        super().__init__()
        self.button = QPushButton("Push for Window")
        self.button.clicked.connect(self.show_new_window)
        self.setCentralWidget(self.button)

    def show_new_window(self, checked):
        w = AnotherWindow()
        w.show()


app = QApplication(sys.argv)
w = MainWindow()
w.show()
app.exec()
```

Clase 15/10/2024

En la clase anterior estuvimos realizando ejemplos hoy terminamos los ejemplos y hay que ponerse con el proyecto

Ejercicios - PRACTICAR EJEMPLOS.

Proyectos - EXPLICAR Y PONER FECHA.

Examen - PONER FECHA.

Ejercicio 1

Crea una pantalla con dos botones en el primer botón debemos que modifique algún otro elemento visual y el segundo otro.

Este ejemplo está realizado qt designer

Solución

Ejemplo2.ui

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>MainWindow</class>
 <widget class="QMainWindow" name="MainWindow">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>548</width>
    <height>411</height>
   </rect>
  </property>
  <property name="windowTitle">
   <string>MainWindow</string>
  </property>
  <widget class="QWidget" name="centralwidget">
   <widget class="QPushButton" name="mibtn">
    <property name="geometry">
     <rect>
      <x>210</x>
      <y>250</y>
      <width>75</width>
      <height>23</height>
     </rect>
    </property>
    <property name="text">
     <string>PushButton</string>
    </property>
   </widget>
   <widget class="QPushButton" name="pushButton">
    <property name="geometry">
     <rect>
      <x>290</x>
      <y>250</y>
      <width>75</width>
      <height>23</height>
     </rect>
    </property>
    <property name="text">
     <string>PushButton</string>
    </property>
   </widget>
   <widget class="QTableWidget" name="tableWidget">
    <property name="geometry">
     <rect>
      <x>140</x>
      <y>10</y>
      <width>256</width>
      <height>111</height>
     </rect>
    </property>
    <row>
     <property name="text">
      <string>New Row</string>
     </property>
    </row>
    <row>
     <property name="text">
      <string>New Row</string>
     </property>
    </row>
    <column>
     <property name="text">
      <string>New Column</string>
     </property>
    </column>
    <column>
     <property name="text">
      <string>New Column</string>
     </property>
    </column>
   </widget>
  </widget>
  <widget class="QMenuBar" name="menubar">
   <property name="geometry">
    <rect>
     <x>0</x>
     <y>0</y>
     <width>548</width>
     <height>22</height>
    </rect>
   </property>
  </widget>
  <widget class="QStatusBar" name="statusbar"/>
 </widget>
 <resources/>
 <connections/>
</ui>
```

El código que controla la pantalla:

```
#importamos las librerias necesarias
import sys
from PyQt6 import QtWidgets, uic

from PyQt6.QtWidgets import (QApplication, QDialog, QMainWindow, QPushButton,
                             QVBoxLayout, QLabel, QDialogButtonBox, QMessageBox)

#Carga la interfaz gráfica y conecta los botones
class Ventana(QtWidgets.QMainWindow):
    '''Esta es la clase principal'''
    #Inicializamos la ventana y conectamos los botones
    def __init__(self, padre=None):
        #Inicializa la ventana
        QtWidgets.QMainWindow.__init__(self, padre)
        uic.loadUi("ejemplo2.ui",self) #Lee el archivo de QtDesigner

        self.setWindowTitle("Ejemplo") #Titulo de la ventana

        #Conectar botón a función
        self.pushButton.clicked.connect(self.funcion)
        self.mibtn.clicked.connect(self.funcion2)

    def funcion(self):
        print("ejemplo")

    def funcion2(self):
        msgBox = QMessageBox()
        msgBox.setText("Ejemplo de qt designer")
        msgBox.exec()


# se crea la instancia de la aplicación
app = QtWidgets.QApplication(sys.argv)
# se crea la instancia de la ventana
miVentana = Ventana()
# se muestra la ventana
miVentana.show()
# se entrega el control al sistema operativo
sys.exit(app.exec())
```

Ejercicio 2: Crea dos ventanas con qt designer que registre información y que con un botón podamos mostrar otra

Ejemplo: Cómo mostrar de una ventana otra.

Hay dos clases cada clase representa cada ventana y desde la ventana principal hay un botón que llama a la nueva ventana.
Modifica el siguiente código para conseguir mostrar tus ventanas diseñadas con qt designer.

```
from PyQt6.QtWidgets import QApplication, QMainWindow, QPushButton, QLabel, QVBoxLayout, QWidget

import sys


class AnotherWindow(QWidget):
    """
    This "window" is a QWidget. If it has no parent, it
    will appear as a free-floating window as we want.
    """
    def __init__(self):
        super().__init__()
        layout = QVBoxLayout()
        self.label = QLabel("Another Window")
        layout.addWidget(self.label)
        self.setLayout(layout)


class MainWindow(QMainWindow):

    def __init__(self):
        super().__init__()
        self.button = QPushButton("Push for Window")
        self.button.clicked.connect(self.show_new_window)
        self.setCentralWidget(self.button)

    def show_new_window(self, checked):
        w = AnotherWindow()
        w.show()


app = QApplication(sys.argv)
w = MainWindow()
w.show()
app.exec()
```

PROYECTO

entrega del proyecto
idea de proyecto
tiene que tener varias pantallas
con componentes visuales

EXAMEN

25/10/2024

Criterios de evluación:

- a) Se han analizado las herramientas y librerías disponibles para la generación de interfaces gráficos. 12,5%
- b) Se ha creado un interfaz gráfico utilizando las herramientas de un editor visual. 12,5%
- c) Se han utilizado las funciones del editor para ubicar los componentes del interfaz. 12,5%
- d) Se han modificado las propiedades de los componentes para adecuarlas a las necesidades de la aplicación. 12,5%
- e) Se ha analizado el código generado por el editor visual. 12,5%
- f) Se ha modificado el código generado por el editor visual. 12,5%
- g) Se han asociado a los eventos las acciones correspondientes. 12,5%
- h) Se ha desarrollado una aplicación que incluye el interfaz gráfico obtenido. 12,5%

PREPARAR LAS MÁQUINAS PARA REALIZAR PRUEBA

Sesión 1: Introducción a JavaFX

Objetivos:

- Comprender qué es JavaFX y cómo se utiliza para crear interfaces gráficas.
- Diferenciar entre JavaFX, Swing y AWT.
- Familiarizarse con la arquitectura básica de JavaFX.
- Configurar el entorno de desarrollo y realizar una instalación básica del JavaFX SDK.
- Crear un proyecto simple que muestre una ventana con el texto "Hola, JavaFX".

Contenido

1. ¿Qué es JavaFX?

- **Definición**: JavaFX es una plataforma de desarrollo de aplicaciones gráficas (GUI) para aplicaciones de escritorio y aplicaciones basadas en internet. Permite crear interfaces de usuario modernas con componentes visuales avanzados, gráficos 2D y 3D, y animaciones.
- **Historia**: Anunciada como el sucesor de Swing para crear aplicaciones gráficas en Java. Originalmente fue lanzado como una biblioteca externa, pero con Java 8, se integró oficialmente como parte de la plataforma Java SE.

---

2. Comparación entre JavaFX, Swing y AWT

Características	AWT	Swing	JavaFX
Fecha de lanzamiento	1995	1997	2008
Diseño	Basado en los componentes nativos del sistema operativo.	No depende de los componentes nativos, tiene su propia implementación.	No depende de componentes nativos. Usa un motor gráfico avanzado.
Gráficos	Limitados a gráficos 2D.	Limitados a gráficos 2D.	Soporte para gráficos 2D y 3D.
Estilización	Difícil de personalizar.	Personalización limitada, se pueden usar Look and Feel.	Totalmente personalizable con CSS.
Animaciones	No soportadas.	No soportadas.	Soporte para animaciones.
Uso actual	Obsoleto.	Aún en uso, pero recomendado solo para mantenimiento.	Plataforma moderna para aplicaciones gráficas.

Conclusión: JavaFX es la evolución natural, con soporte para gráficos avanzados, animaciones, y un mayor nivel de personalización, lo que lo convierte en una opción preferida para aplicaciones modernas.

---

3. Ventajas y desventajas de JavaFX

- **Ventajas**:
  - **Personalización con CSS**: Se puede cambiar el estilo y la apariencia de los componentes mediante hojas de estilo, similar al desarrollo web.
  - **Gráficos y Animaciones**: JavaFX permite la creación de gráficos 2D/3D avanzados y animaciones fluidas.
  - **FXML**: Una forma declarativa de diseñar interfaces de usuario, separando la lógica de la interfaz.
  - **Compatibilidad**: Funciona en múltiples plataformas (Windows, macOS, Linux).
  - **Soporte para multimedia**: Permite integrar vídeo, audio y gráficos interactivos.
- **Desventajas**:
  - **Curva de aprendizaje**: Es más complejo que Swing para los principiantes.
  - **Compatibilidad con Swing**: A pesar de la interoperabilidad, mezclar Swing con JavaFX puede ser complicado.
  - **Adopción**: Aunque es una tecnología moderna, no tiene tanta adopción como frameworks web.

---

4. Arquitectura básica de JavaFX

JavaFX utiliza una arquitectura basada en tres elementos principales:

1. **Stage (Escenario)**:
   - Es la ventana principal o contenedor donde se muestra el contenido.
   - Un Stage puede ser el "primary stage" (ventana principal) o se pueden crear múltiples stages.
2. **Scene (Escena)**:
   - Un Stage contiene una Scene.
   - La Scene es donde se coloca el contenido, como botones, textos y otros componentes.
3. **Node (Nodo)**:
   - Los componentes dentro de una Scene (botones, etiquetas, imágenes) son llamados Nodes.
   - Todos los elementos visuales son instancias de Node, lo que incluye tanto elementos de interfaz como gráficos.

Diagrama:

```
Stage (Ventana)
   └── Scene (Contenedor principal)
         └── Node (Elementos visuales, por ejemplo, Button, Label)
```

---

5. Configuración del entorno de desarrollo

Actividad Práctica: Crear un Proyecto Básico en JavaFX

Objetivo: Crear una ventana básica que muestre el mensaje "Hola, JavaFX".

Pasos:

1. **Crear el proyecto**
   https://gluonhq.com/products/scene-builder/#download
   Create java projects
   añadir librerias
   vmoptions
2. **Escribir el código** para generar una ventana con texto. Aquí tienes un ejemplo básico:

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.stage.Stage;

public class HolaJavaFX extends Application {

    @Override
    public void start(Stage stage) {
        // Crear un label con el texto
        Label mensaje = new Label("Hola, JavaFX");

        // Crear la escena con el Label
        Scene scene = new Scene(mensaje, 300, 200);

        // Configurar el stage (ventana)
        stage.setTitle("Mi primera aplicación JavaFX");
        stage.setScene(scene);
        stage.show(); // Mostrar la ventana
    }

    public static void main(String[] args) {
        launch(args); // Iniciar la aplicación
    }
}
```

3. **Ejecutar el proyecto**: Después de configurar las VM Options correctamente, deberías ver una ventana emergente con el mensaje "Hola, JavaFX".

Conclusión y Reflexión:

- **Discusión en clase**: Reflexionar sobre la experiencia con JavaFX.

Clase: Sesión 2 - Estructura de una Aplicación JavaFX

Objetivos:

- Comprender los componentes principales de una aplicación JavaFX.
- Implementar una ventana básica con un layout y botones utilizando contenedores de JavaFX.
- Introducir el uso de CSS para la personalización visual.

---

1. Introducción (15 minutos)

Repaso rápido de conceptos básicos

¿Qué es JavaFX?

JavaFX es una librería para desarrollar interfaces gráficas modernas en Java. Los componentes esenciales de una aplicación JavaFX son:

- `Stage`: Representa la ventana de la aplicación.
- `Scene`: Representa el contenido dentro de la ventana.
- `Nodes`: Elementos que forman parte de la interfaz (botones, campos de texto, etc.).

Presentación de la clase `Application`

La clase `Application` es la base para cualquier aplicación JavaFX y tiene un ciclo de vida que incluye:

- `start(Stage primaryStage)`: Método principal donde se define la interfaz.
- `init()`: Se ejecuta antes de `start()`, útil para inicializaciones.
- `stop()`: Se ejecuta cuando se cierra la aplicación.

Ejemplo de código básico:

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;

public class MainApp extends Application {
    @Override
    public void start(Stage primaryStage) {
        primaryStage.setTitle("Mi Primera App JavaFX");
        VBox root = new VBox();
        Scene scene = new Scene(root, 300, 250);
        primaryStage.setScene(scene);
        primaryStage.show();
    }

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

Ciclo de vida de una aplicación JavaFX:

1. `launch(args)`: Lanza la aplicación.
2. Se llama a `start()` para crear la ventana y mostrarla.
3. La ventana se muestra con `primaryStage.show()`.

---

2. Desarrollo de la Interfaz Gráfica

Introducción a Contenedores (Layouts)

Los **contenedores** en JavaFX organizan los elementos de la interfaz gráfica.

- **VBox**: Coloca los nodos uno debajo de otro.
- **HBox**: Coloca los nodos uno al lado de otro.
- **BorderPane**: Distribuye los nodos en top, bottom, left, right, center.
- **GridPane**: Organiza los nodos en una cuadrícula.

Ejemplo con VBox:

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;

public class VBoxExample extends Application {
    @Override
    public void start(Stage primaryStage) {
        primaryStage.setTitle("Ejemplo VBox");

        VBox vbox = new VBox();
        Button btn1 = new Button("Botón 1");
        Button btn2 = new Button("Botón 2");

        vbox.getChildren().addAll(btn1, btn2);

        Scene scene = new Scene(vbox, 300, 200);
        primaryStage.setScene(scene);
        primaryStage.show();
    }

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

Actividad práctica 1:

- Crear una ventana con dos botones en un VBox.
- **Instrucciones**: Añadir dos botones y organizarlos verticalmente.

---

3. Introducción al CSS en JavaFX

Introducción al uso de CSS

Los archivos CSS se pueden usar en JavaFX para aplicar estilos a los elementos visuales.

Ejemplo de archivo CSS:

```
/* style.css */
.boton-rojo {
    -fx-background-color: red;
    -fx-text-fill: white;
}
```

Vincular el archivo CSS con JavaFX:

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;

public class CSSExample extends Application {
    @Override
    public void start(Stage primaryStage) {
        primaryStage.setTitle("Ejemplo CSS");

        VBox vbox = new VBox();
        Button btn1 = new Button("Botón 1");
        Button btn2 = new Button("Botón 2");

        btn1.getStyleClass().add("boton-rojo");

        vbox.getChildren().addAll(btn1, btn2);

        Scene scene = new Scene(vbox, 300, 200);
        scene.getStylesheets().add("style.css");

        primaryStage.setScene(scene);
        primaryStage.show();
    }

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

Actividad práctica 2:

- Crear un archivo `style.css` y aplicar un estilo personalizado a los botones.
- **Instrucciones**: Crear un archivo CSS que cambie el color de fondo y texto del botón.

Tarea:

- **Desafío opcional**: Añadir un tercer botón y cambiar su estilo con CSS (ej. cambiar tamaño de fuente, color de texto).

Material necesario:

- Ordenadores con Java y JavaFX instalados.
- Un IDE configurado con soporte para JavaFX.
- Acceso a internet para consultar documentación (opcional).

Recursos adicionales:

- Documentación oficial de JavaFX (https://openjfx.io/)
- Tutoriales de JavaFX en línea para reforzar conceptos.