EXAMEN PARCIAL INGENIERÍA SISTEMAS SOFTWARE BASADOS EN CONOCIMIENTO DÍA 23/03

Ejercicio de examen utilizando MVC

La arquitectura Modelo-Vista-Controlador (MVC) es un patrón de diseño de software que se utiliza comúnmente para estructurar aplicaciones de software que tienen una interfaz de usuario. El patrón MVC separa la aplicación en tres componentes principales:

1. Modelo: representa la estructura de datos y la lógica de negocios de la aplicación.
2. Vista: se encarga de la presentación de los datos al usuario.
3. Controlador: actúa como intermediario entre la vista y el modelo.

En este caso el ejercicio ha consistido en la creación de dos tablas unido a una serie de botones en la parte inferior de la ventana.

Lo primero que se ha realizado ha sido el modelo de la aplicación, en el cual se han añadido todas las funciones requeridas por el problema, como son:

- Añadir una fila a la tabla de la izquierda:

```
#Añade un alumno a la tabla de alumnos
def addAlumnos(self):
    row_count = self.table1.rowCount()
    self.table1.insertRow(row_count)
```

- Suprimir la fila de la tabla de la izquierda que esté seleccionada:

```
#Elimina un alumno seleccionado de la tabla de alumnos
def deleteAlumnos(self):
    row = self.table1.currentRow()
    self.table1.removeRow(row)
```

- Ordenar los datos de los alumnos por nombre y apellidos en la tabla de la izquierda:

```
#Ordena la tabla de alumnos por nombre
def ordAlumnos(self):
    self.table1.sortItems(0, 0)
```

- Seleccionar los alumnos aprobados de la tabla de la izquierda y pasar esa información a la tabla de la derecha:

```
#Selecciona los alumnos aprobados y los añade a la tabla de aprobados
def selectAlumnos(self):
    for row in range(self.table2.rowCount()-1, -1, -1):
        self.table2.removeRow(row)

    for row in range(self.table1.rowCount()):
        nota = int(self.table1.item(row, 1).text())
        if nota >= 5:
            fila = []
            for column in range(self.table1.columnCount()):
                fila.append(self.table1.item(row, column).clone())
            self.table2.insertRow(self.table2.rowCount())
            for column, item in enumerate(fila):
                self.table2.setItem(self.table2.rowCount()-1, column, item)
```

- Guardar los datos de la tabla de la izquierda en un archivo de texto:

```
#Guarda la tabla de alumnos en un fichero de texto
def saveAs(self):
    home = str(Path.home())
    newFileName, _ = QFileDialog.getSaveFileName(self, "Save File", home, "Text Files (*.txt)")

    if newFileName:
        file = QFile(newFileName)
        file.open(QFile.WriteOnly | QFile.Text)
        stream = QTextStream(file)
        for row in range(self.table1.rowCount()):
            for column in range(self.table1.columnCount()):
                item = self.table1.item(row, column)
                if item is not None:
                    stream << item.text() << "\t"
            stream << "\n"
        file.close()
```

- Abrir un archivo de texto y pasar la información a la tabla de la izquierda:

```
#Abre un fichero de texto y lo carga en la tabla de alumnos
def openFile(self):
    home = str(Path.home())
    filename, _ = QFileDialog.getOpenFileName(self, "Open File", home, "Text Files (*.txt)")

    if filename:
        with open(filename, 'r') as file:
            lines = file.readlines()
            data = [line.strip().split("\t") for line in lines]
            nRows = len(data)
            nColums = len(data[0])
            self.table1.setRowCount(nRows)
            self.table1.setColumnCount(nColums)
            for row in range(nRows):
                for colum in range(nColums):
                    cell = QTableWidgetItem(data[row][colum])
                    self.table1.setItem(row, colum, cell)
```

Además, se ha definido una función para restringir los valores insertados en la columna de notas, siendo los valores permitidos enteros entre 0 y 10. Si se inserta cualquier otro carácter o valor, en la tabla se escribirá un 0:

```
#Controla que los valores introducidos en la tabla de alumnos sean correctos
def cell_changed(self, row, column):
    item = self.table1.item(row, column)
    if item is None:
        return
    value = item.text()
    try:
        value = int(value)
    except ValueError:
        self.table1.setItem(row, column, QTableWidgetItem("0"))
        return
    if value < 0 or value > 10:
        self.table1.setItem(row, column, QTableWidgetItem("0"))
        return
    item.setText(str(value))
```

Todo esto se encuentra en el archivo:
https://www.uco.es/~i02rofls/EjercicioPractico/modelTableCalifications.py

A continuación, se han creado las vistas de la aplicación, utilizando una ventana principal y añadiendo las dos tablas requeridas sin filas iniciales, así como los diferentes botones de las opciones.

```
#Se crean ambas tablas
self.table1 = QTableWidget(0, 2)
self.table1.setHorizontalHeaderLabels(['Nombre', 'Nota'])
self.table1.horizontalHeader().setSectionResizeMode(0, QHeaderView.Stretch)
self.table1.horizontalHeader().resizeSection(1, 100)

self.table2 = QTableWidget(0, 2)
self.table2.setHorizontalHeaderLabels(['Nombre', 'Nota'])
self.table2.setEditTriggers(QTableWidget.NoEditTriggers)
self.table2.horizontalHeader().setSectionResizeMode(0, QHeaderView.Stretch)
self.table2.horizontalHeader().resizeSection(1, 100)

#Se crean los botones
self.ordButton = QPushButton("Ordenar")
self.selectButton = QPushButton("Seleccionar Aprobados")
self.addButton = QPushButton("Añadir")
self.deleteButton = QPushButton("Borrar")
self.saveButton = QPushButton("Guardar")
self.openButton = QPushButton("Abrir")
```

Asimismo, se ha restringido la tabla de la derecha para que no sea posible añadir contenido.

Tras ello, se han definido las cuadrículas de botones y se ha organizado todo:

```
#Se crea la cuadrícula de botones y se añaden
grid = QGridLayout()
grid.setSpacing(10)
grid.addWidget(self.ordButton, 2, 0)
grid.addWidget(self.selectButton, 2, 1)

grid2 =QGridLayout()
grid2.setSpacing(10)
grid2.addWidget(self.addButton, 2, 2)
grid2.addWidget(self.deleteButton, 2, 3)

grid3 = QGridLayout()
grid3.setSpacing(10)
grid3.addWidget(self.saveButton, 2, 3)
grid3.addWidget(self.openButton, 2, 4)

#Se crea la cuadrícula principal y se añaden las tablas y la cuadrícula de botones
layout = QGridLayout()
layout.addWidget(QLabel("Lista de Notas"), 0, 0)
layout.addWidget(QLabel("Lista de Aprobados"), 0, 1)
layout.addWidget(self.table1, 1, 0)
layout.addWidget(self.table2, 1, 1)
layout.addLayout(grid, 2, 0)
layout.addLayout(grid2, 3, 0)
layout.addLayout(grid3, 2, 1)

self.setLayout(layout)
```

Finalmente, se han conectado los botones a los diferentes eventos del controlador:

```
#Se conectan los botones con los eventos
self.ordButton.clicked.connect(self.ordAlumnos)
self.selectButton.clicked.connect(self.selectAlumnos)
self.addButton.clicked.connect(self.addAlumnos)
self.deleteButton.clicked.connect(self.deleteAlumnos)
self.saveButton.clicked.connect(self.saveAs)
self.openButton.clicked.connect(self.openFile)

#Se conecta la tabla con el evento para no permitir escrbir en la columna de notas un valor distinto de [0, 10]
self.table1.cellChanged.connect(self.cell_changed)
```

Esta vista se encuentra en el archivo:
https://www.uco.es/~i02rofls/EjercicioPractico/viewTableCalifications.py

Hecho esto, se codifica el controlador, en el cual se ha añadido la conexión de los botones de la vista con las acciones indicadas en el modelo, conectando así las distintas acciones que puede realizar la aplicación:

```
# Eventos de la tabla de calificaciones
def eventOrdAlumnos(self):
    mapp.ordAlumnos(self)

def eventSelectAlumnos(self):
    mapp.selectAlumnos(self)

def eventAddAlumnos(self):
    mapp.addAlumnos(self)

def eventDeleteAlumnos(self):
    mapp.deleteAlumnos(self)

def eventSaveAs(self):
    mapp.saveAs(self)

def eventOpenFile(self):
    mapp.openFile(self)

def eventCellChanged(self, row, column):
    mapp.cell_changed(self, row, column)
```

Este controlador se encuentra en:
https://www.uco.es/~i02rofls/EjercicioPractico/controllerTableCalifications.py

Finalmente, se codifica el fichero de aplicación que iniciará todo lo anterior y el cual se encuentra:
https://www.uco.es/~i02rofls/EjercicioPractico/appTableCalifications.py

```
app = QtWidgets.QApplication(sys.argv) #Crea una aplicación
form = view.TableCalificationsDlg()   #Crea una instancia del formulario
sys.exit(app.exec_())   #Se inicia la aplicación y se espera eventos
```