UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
FACULTAD DE INGENIERÍA
ESCUELA DE CIENCIAS Y SISTEMAS
SEGUNDO SEMESTRE 2022

LFP B+

MANUAL TÉCNICO

PRÁCTICA NO.1

Nombre: Sergio André Lima Corado
Carné: 202100154
Auxiliar: Diego Andrés Obín Rosales

Fecha: 21/08/2022

Índice

PARADIGMAS

Programación orientada a Objetos:

El principal paradigma utilizado en esta aplicación es el paradigma de programación orientado a objetos, ya que se utilizaron distintas clases para facilitar el manejo de los cursos y de la GUI, de esta manera se podía establecer la estructura de un curso y poder acceder a los distintos atributos con los que cuenta

Programación Procedimental

Otro paradigma utilizado en esta aplicación fue el paradigma de programación procedimental, este paradigma fue utilizado en muchas de las funciones definidas en la aplicación como las funciones definidas para realizar el cálculo de los créditos.

CLASE CURSO

Se definió la clase curso, la cual forma la estructura en la que un curso será almacenado tras leerse del archivo de texto:

```python
class Curso:
    def __init__(self, codigo, nombre, prerequisitos, opcionalidad, semestre, creditos, estado):
        self.codigo = codigo
        self.nombre = nombre
        self.prerequisitos = prerequisitos.split(';')
        if opcionalidad == '1':
            self.obligatorio = True
        else:
            self.obligatorio = False
        self.semestre = int(semestre)
        self.creditos = int(creditos)
        self.estado = int(estado)
```

Las funciones disponibles son para editar la información del curso y para poder imprimir en un string, los datos ingresados del estado y opcionalidad, es decir traducir el carácter que representa el estado hacia un String que el usuario pueda entender e interpretar en la GUI.

```python
def editarCurso(self, codigo, nombre, prerequisitos, opcionalidad, semestre, creditos, estado):
    self.codigo = codigo
    self.nombre = nombre
    self.prerequisitos = prerequisitos.split(';')
    if opcionalidad == '1':
        self.obligatorio = True
    else:
        self.obligatorio = False
    self.semestre = int(semestre)
    self.creditos = int(creditos)
    self.estado = int(estado)
```

```
def printObligatorio(self):
    if self.obligatorio:
        return 'Obligatorio'
    else:
        return 'Opcional'

def printEstado(self):
    if self.estado == 0:
        return 'Aprobado'
    elif self.estado == 1:
        return 'Cursando'
    else:
        return 'Pendiente'

def printPreRequisitos(self):
    if self.prerequisitos is None:
        return 'Ninguno'
    else:
        txt = ", ".join(self.prerequisitos)
        return txt
```

CLASE DATABASE

Esta clase básicamente crea una ‘base de datos’ en dónde se almacenan los arrays en que se almacenan los cursos, pues los atributos de la clase es un array que almacenará los cursos y otro con los códigos de los cursos.

```
class Database:
    def __init__(self):
        self.cursos = []
        self.codigosCursos = []
```

La primera función de esta clase es la que genera la carga masiva a partir de un archivo de texto. Ya que los datos son almacenados dentro de los arrays.

```
def cargaMasiva(self, ruta):
    archivo = open(ruta, 'r', encoding='utf8')
    lineas = archivo.readlines()
    print(lineas)
    for linea in lineas:
        fila = linea.split(',')
        curso = Curso(fila[0], fila[1], fila[2], fila[3], fila[4], fila[5], fila[6])
        self.agregarCurso(curso)
        print(fila)
```

Dicha función también recae en la función que agrega un curso a la base de datos.

```
def agregarCurso(self, curso):
    if not (curso.codigo in self.codigosCursos):
        self.cursos.append(curso)
        self.codigosCursos.append(curso.codigo)
    else:
        cursoRegistrado = self.getCurso(curso.codigo)
        cursoRegistrado.sobreescribirCurso(curso.codigo, curso.nombre, curso.prerequisitos,
                                           curso.semestre, curso.creditos, curso.estado)
```

También contiene funciones que navegan en los arrays, buscando y eliminando elementos de la base de datos.

```python
def getCurso(self, codigo):
    if codigo in self.codigosCursos:
        for curso in self.cursos:
            if curso.codigo == codigo:
                return curso
    return None

def eliminarCurso(self, codigo):
    if self.getCurso(codigo):
        curso = self.getCurso(codigo)
        self.cursos.remove(curso)
        self.codigosCursos.remove(codigo)
        return True
    else:
        return None
```

Por último, las funciones que generan el conteo de los créditos y lo retornan.

```python
def getCreditosAprobados(self):
    creditos = 0
    for curso in self.cursos:
        if curso.obligatorio and curso.estado == 0:
            creditos += curso.creditos
    return creditos

def getCreditosCursando(self):
    creditos = 0
    for curso in self.cursos:
        if curso.estado == 1:
            creditos += curso.creditos
    return creditos

def getCreditosPendientes(self):
    creditos = 0
    for curso in self.cursos:
        if curso.obligatorio and curso.estado == -1
            creditos += curso.creditos
    return creditos
```

```python
def getCreditosHastaSemestreN(self, n):
    creditos = 0
    for curso in self.cursos:
        if curso.obligatorio and curso.semestre <= int(n):
            creditos += curso.creditos
    return creditos

def getCreditosSemestre(self, n):
    creditos = [0, 0]  # [Aprobados, Pendientes]
    for curso in self.cursos:
        if curso.semestre == int(n):
            if curso.estado == 0:
                creditos[0] += curso.creditos
            elif curso.estado == -1:
                creditos[1] += curso.creditos
    return creditos
```

Clase GUI

Esta clase contiene cada una de las ventanas que componen la GUI de la aplicación. Todas las clases contienen la misma lógica basada en ventanas con la librería de Tkinter. Lo que varía entre las clases son las funciones, que llaman a distintas clases y funciones.

La estructura de cada clase es la siguiente:

Primero se crea la clase y al iniciarla construye una ventana con las características que se configuran en el constructor, además de tener una función que centra la ventana en la pantalla.

```
class PantallaPrincipal:
    def __init__(self):
        self.ventana = Tk()
        self.ventana.resizable(False, False)
        self.ventana.title('Práctica 1 - LFP B+')
        self.Centrar(self.ventana, 1200, 800)
        self.ventana.geometry('1200x800')
        self.ventana.configure(bg='#191935')
        self.Ventana()

    def Centrar(self, r, ancho, alto):
        altura_pantalla = r.winfo_screenheight()
        ancho_pantalla = r.winfo_screenwidth()

        x = (ancho_pantalla // 2) - (ancho // 2)
        y = (altura_pantalla // 2) - (alto // 2)
        r.geometry(f'+{x}+{y}')
```

Posteriormente se define en la clase ventana todos los widgets con los que cuenta la ventana, es decir todos los labels, entrys, botones, entre otros. Y se define la acción que realiza cada botón. Al finar se añade el mainloop para que la ventana espere un evento y no se cierre de inmediato.

```
def Ventana(self):
    self.frame = Frame(height=700, width=1100)
    self.frame.config(bg='#2D4263')
    self.frame.pack(padx=0, pady=50)
    Label(self.frame, text="Curso: Lenguajes Formales y de Programación\n"
                           "Nombre: Sergio André Lima Corado\nCarné: 202100154",
          font=('Roboto', 35), fg='#EEEEEE', bg='#C84B31',
          width=40).place(x=5, y=50)
    Button(self.frame, text="Cargar Archivo", font=('Roboto', 25), fg='#000000',
           bg='#ECDBBA', width=20, command=self.IrCargarArchivos).place(x=350, y=300)
    Button(self.frame, text="Gestionar Cursos", font=('Roboto', 25), fg='#000000',
           bg='#ECDBBA', width=20, command=self.IrGestionarCursos).place(x=350, y=400)
    Button(self.frame, text="Conteo de Créditos", font=('Roboto', 25), fg='#000000',
           bg='#ECDBBA', width=20, command=self.IrCreditos).place(x=350, y=500)
    Button(self.frame, text="Salir", font=('Roboto', 25), fg='#000000',
           bg='#ECDBBA', width=20, command=self.Salir).place(x=350, y=600)
    self.frame.mainloop()
```

Por último están las funciones que ejecutan los botones que permiten abrir otras ventanas o llaman funciones de otra clase.

```
def IrCargarArchivos(self):
    self.ventana.destroy()
    PantallaCargarArchivos()

def IrCreditos(self):
    self.ventana.destroy()
    PantallaCreditos()

def IrGestionarCursos(self):
    self.ventana.destroy()
    PantallaGestionarCursos()

def Salir(self):
    self.ventana.destroy()
```