# Practica 1 de Lenguajes Formales y de Programación

## Manual Técnico

### Paradigmas utilizados:

Para esta práctica se estuvo manejando el paradigma de la programación orientada a objetos, esto con el fin de hacer uso de clases, en las que se pudiera guardar toda la información que el usuario necesitara cargar en memoria. Para acceder de forma particular a cada uno de los datos de la clase se utilizaron métodos Gets y Sets, de manera que fuera mucho mas simple acceder a ellos y no de manera encapsulada llamado un método cada vez que se quiera hacer uso de los atributos.

```python
class peliculas:

    def __init__(self, nombrePelicula, actores, anoPelicula, generoPelicula):
        self.nombrePelicula = nombrePelicula
        self.actores = actores
        self.anoPelicula = anoPelicula
        self.generoPelicula = generoPelicula


    # Metodos Get
    def get_pelicula(self):
        return self.nombrePelicula

    def get_actores(self):
        return self.actores

    def get_anoPelicula(self):
        return self.anoPelicula

    def get_generoPelicula(self):
        return self.generoPelicula


    # Metodos SET
    def set_pelicula(self, nombrePelicula):
        self.nombrePelicula = nombrePelicula

    def set_actores(self, actores):
        self.actores = actores

    def set_anoPelicula(self, anoPelicula):
        self.anoPelicula = anoPelicula

    def set_generoPelicula(self, generoPelicula):
        self.generoPelicula = generoPelicula
```

### Librerías y herramientas Adicionales:

- Librería OS: esta librería se utilizo mas que nada para limpiar la consola después de acceder a los menús, de manera que el resultado final del programa sea mucho más entendible y estético

- Graphviz: esta herramienta permite crear grafos de manera simple y fácil de entender, fue utilizada para crear los reportes finales en los que se relacionan los actores presentes con las películas en las que participaron

**Carga de Archivo:**

Para la carga de archivos es necesario que el usuario ingrese la ruta absoluta donde se encuentra el archivo, una vez le da enter el programa se dirige a la ruta del archivo y lee el mismo

```
# Se abre el archivo para poder unicamente leerlo
ruta = input("Ingrese la ruta ABASOLUTA del arhivo: ")
archivo = open(ruta, "r")
lineas = archivo.readlines()
```

El programa se encarga de leer línea por línea y de separar cada uno de los elementos que estén separados por punto y coma en cada línea, para esto se utiliza la función split(;)

Para esto se utiliza un bucle for que recorra cada línea en búsqueda de nuevos elementos, en caso de que el nombre de la película coincida con el nombre de una película cargada en memoria con anterioridad, la nueva película sustituirá a la ya cargada, y la nueva se cargara 1 sola vez, esto con el fin de evitar la duplicación

Para este método se utilizan los sets de manera que puedan asignar el valor duplicado una sola vez.

```
for y in lista:
    if y.get_pelicula() == temporal_nombrePelicula:

        lista[lista.index(y)].set_pelicula(temporal_nombrePelicula)
        lista[lista.index(y)].set_actores(temporal_actores)
        lista[lista.index(y)].set_anoPelicula(temporal_anoPelicula)
        lista[lista.index(y)].set_generoPelicula(temporal_generoPelicula)
        repetido = True
        contadorRepetidos += 1
```

Para ir guardando los datos de la clase, se hace uso de una lista general.

```
# Hace uso de la clase peliculas y agrega los atributos
datos = peliculas(temporal_nombrePelicula, temporal_actores, temporal_anoPelicula ,temporal_generoPelicula)

# Agrega al final de la lista los datos
lista.append(datos)
contadorNoRepetidos += 1
```

En donde atreves de la función append, va añadiendo los datos de la clase al final de la lista.

### Menú Principal y Sub Menús

Para el menú principal se hace uso de un ciclo while, que se detiene en caso que la opción seleccionada sea la de salir, en ese caso el programa entra en el case de salida, que ejecuta un break para parar el bucle especifico.

```
while opcion != 5:
    #Limpiar la pantalla
    os.system ("cls")
    submenu = 0

    print("------------------   MENU   ------------------\n")
    print("""
    1.  Cargar archivo de entrada
    2.  Gestion de películas
    3.  Filtrado
    4.  Gráfica
    5.  Salir
    """)
    opcion = int(input("\n Ingrese el numero de la opcion y presione enter: "))
```

```
        case 5:
            os.system ("cls")
            input("Gracias por usar este programa !_! ")
            break
```

### Muestreo de Datos:

Se hace uso de un bucle for que recorra cada una de las listas de datos disponibles en la lista de películas, el bucle imprime el valor de la variable x + la función get que hace referencia al valor que se quiere llamar

```
case 1:
    os.system ("cls")
    # Recorre la lista entera de peliculas para imprimir los datos de cada una
    for x in listaDePeliculas:
        print("\nNombre del la Pelicula: ", x.get_pelicula(),
            "\nActores: ", x.get_actores() ,"\nAño de Estreno: ",
            x.get_anoPelicula() ,"\nGenero de la Pelicula: ",
            x.get_generoPelicula())

    input("\nPresiona Enter para continuar")
```

**Filtrado de Datos:**

Para la opción de filtrado se muestra la lista de películas, años y géneros disponibles en memoria, para hacer esto de manera más dinámica y más fácil para el usuario se crearon funciones, donde se guardan listas temporales de películas, años y géneros, una vez el usuario ingresa la opción que desea ver, el programa compara cada uno de los elementos presentes en la lista temporal con la lista maestra de películas, para así mostrar los elementos donde coincida lo que se busca.

```
    os.system ("cls")
    contador = 1
    find = False
    listaTemp_Actores = []
    listaTemp_Peliculas = []


    # Llama a la funcion actores que guarda en una lista todos los actores presentes en memoria
    actores(listaDePeliculas, listaTemp_Actores)

    imprimirFiltrado(listaTemp_Actores, contador)

    # Guarda el nombre del actor
    actor  = int(input("\nIngrese la opcion que desea ver: "))
    actor = listaTemp_Actores[actor-1]

    peliculasFilt(listaDePeliculas, listaTemp_Peliculas, actor)
```

```
def actores(listaGeneral, listaEspecifica):
    for x in listaGeneral:
        for y in x.get_actores():
            y = y.strip()
            if listaEspecifica.count(y) == 0:
                listaEspecifica.append(y)

    return listaEspecifica


def peliculasFilt(listaGeneral, listaEspecifica, dato):
    for x in listaGeneral:
        for y in x.get_actores():

            if y.strip() == dato:
                listaEspecifica.append(x.get_pelicula())

    return listaEspecifica
```

Nota: las funciones se crearon con la idea de ahorrar código y de modo que este fuera mas entendible, pero al final debido a que cada uno de los valores que necesitan presentarse son diferentes se crearon varias funciones. Esto se podría arreglar añadiendo mas variables a la función, de modo que dependiendo de la necesidad únicamente cambie las variables y no la función entera

**Creación de Reporte Grafico:**

Para la creación de los nodos de primero se “escribe” en un archivo todas las líneas necesarias para empezar a configurar los nodos con la herramienta graphviz

```
grafo_dot = open("grafos.dot", "w")
grafo_dot.write('digraph { \n')
grafo_dot.write('rankdir = LR \n' )
grafo_dot.write('node[shape=record, fontname="Arial", fontsize=15] \n')
```

En este caso se utiliza una función donde se le pasan la lista de películas y una lista temporal de actores. Se hizo así pensando en que un actor puede aparecer varias veces en diferentes películas y que no puede haber películas repetidas, por eso mismo el principal a no repetirse son los actores

Para eso se implementa un bucle for que recorra la lista temporal de actores, luego otro bucle que recorra la lista general de películas, y un ultimo bucle que recorra la lista de actores de cada película. En caso de que el valor de la variable de lista temporal de actores coincida con la lista de actores presentes en la lista general de películas, este creara el nodo entre el nombre de la película y el nombre del actor.

Para la creación de nodos se utilizó el índice del elemento de cada lista. Debido a que, para la creación de nodos, estos no pueden estar separados por espacios si no los tomaría como nodos independientes.

```
for x in listaTemp_Actores:
    for y in listaDePeliculas:
        for z in y.get_actores():

            if z.strip() == x:
                grafo_dot.write("peli" + str(listaDePeliculas.index(y)) + '->' + "actor"+ str(listaTemp_Actores.index(x)) + '\n')
```

Para colocar el nombre y la información adicional a cada uno de los nodos, fue necesario crear cabeceras. De igual manera se implementan bucles for para aplicar nombres a cada uno los nodos

```
for i in listaTemp_Actores:
    grafo_dot.write('actor' + str(listaTemp_Actores.index(i)) + '[color = black, fillcolor = olivedrab2, style = filled, label="'+ i + '"]\n')

grafo_dot.write('\n\n')

# Cambia el diseño para las peliculas, y agrega la informacion necesaria, como el genero y año
for j in listaDePeliculas:
    grafo_dot.write('peli' + str(listaDePeliculas.index(j)) + '[color = black, fillcolor = orange, style = filled, label="' + j.get_pelicula() +
                    ' | { ' + j.get_anoPelicula() + ' | ' + j.get_generoPelicula() + '}"]\n')

grafo_dot.write('\n\n')
```

Por ultimo se exporta el archivo.dot como formato pdf, donde están detalladas cada una de las conexiones entre película y actor