

MANUAL TÉCNICO

Diana Estefania
Berducido Domingo

202000277

IPC2 – Laboratorio

Aux. Jackeline Benítez

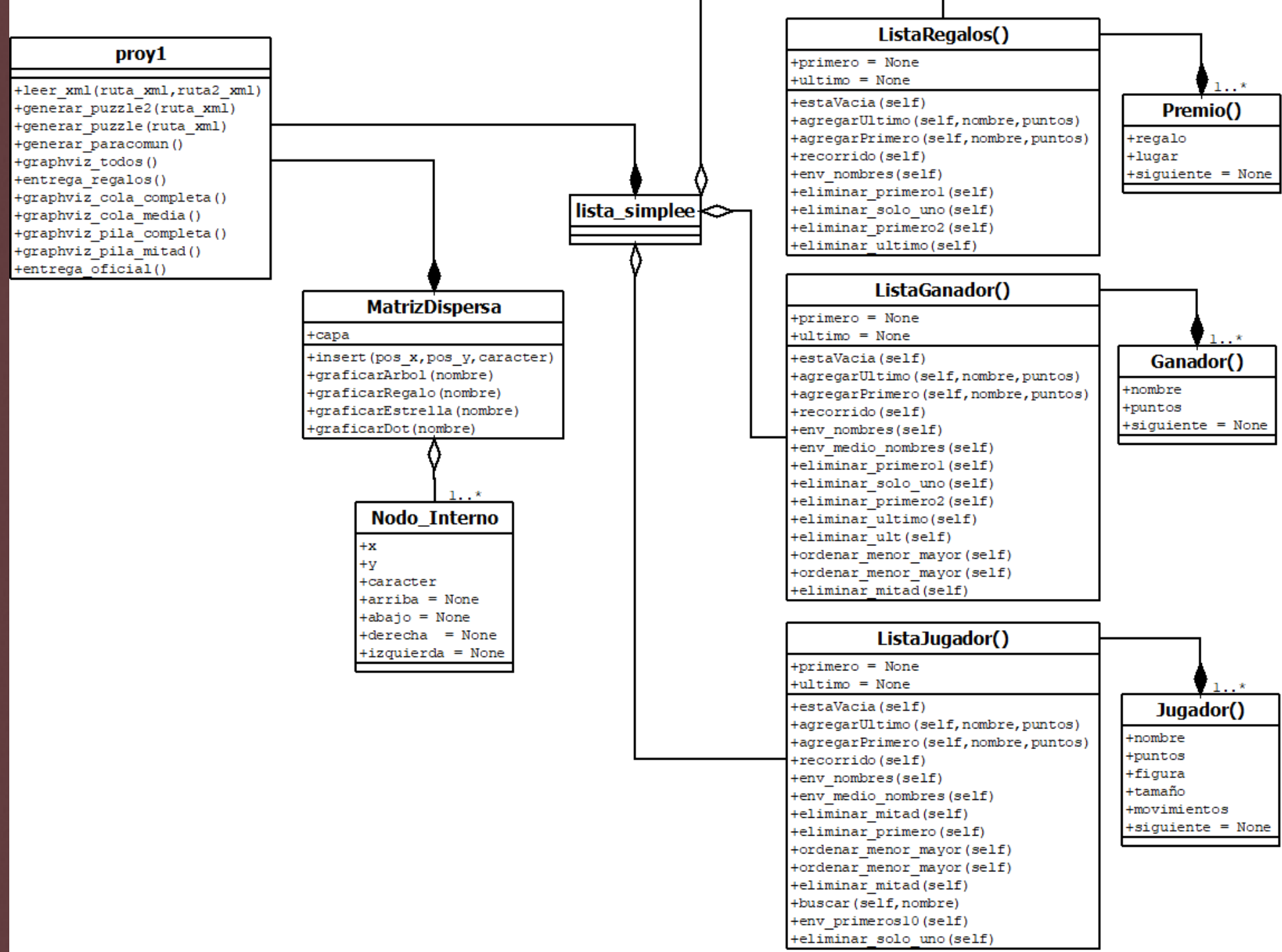
Programación utilizada: POO

- A través de la implementación de estructuras de datos para listas enlazadas simples y matrices dispersas. Estas recopilan objetos y datos (jugadores, ganadores, premios, filas, columnas) que almacena y en su mayoría representa de forma gráfica.

Objetivo del Programa

- El objetivo es obtener dos archivos con estructura XML.
- Se realiza la lectura de los XML y se procede a llenar las estructuras de listas enlazadas simples para ejecutar las colas y pilas.
- El usuario tiene varias opciones: Generar una entrega de regalos simulada en consola, Generar los puzles de todos los jugadores mientras se retiran de su cola reflejado de forma gráfica y Generar una entrega de regalos simulada de forma gráfica donde se observa el inicio de la cola y pila y la mitad de la ejecución.
- El objetivo general es calcular, según los archivos de entrada, los punteos de los jugadores y saber quiénes serían los primeros 10 lugares. A estos primeros lugares se les entrega un premio y esa entrega se refleja como una cola y una pila. Además de poder observar la figura en sus puzles desordenados y ordenados.

Diagrama de Clases



Proy1 (MAIN)

- Posterior a la declaración de varios métodos que ayudan a graficar (siguientes diapositivas) cuenta con el menú de inicio, donde pregunta por los archivos de entrada y solicitan seleccionar una acción:

```
print("-----¡Bienvenid@ al juego de Puzzle!-----")
global ruta
ruta = input("-----Por favor, ingrese el nombre del archivo XML de jugadores: ")
ruta2 = input("-----Por favor, ingrese el nombre del archivo XML de premios: ")
a = leer_xml(f"{ruta}", f"{ruta2}")
regalosss.recorrido()
#graphviz_entrega()
while True:
    print("Ingrese el número de la opción que desea ejecutar:")
    print("1. Simular Entrega de Regalos en Consola.") #listo
    print("2. Ver cola de participantes e ir generando su puzzle mientras se retiran de la cola.")
    print("3. Generar gráficos de Entrega de Regalos.") #listo
    print("4. Salir.")
    opcion = input("Opción: ")
    if opcion == "1": ...
    elif opcion == "2": ...
    elif opcion == "3": ...
    elif opcion == "4": ...
```

Proy1 (MAIN)

- Desde aquí se llama a las otras clases para ejecutar sus funciones. Las funciones en este archivo son:
- Y, todas se encargan de la parte de traer datos de las listas para graficarlas o mostrarlas en consola.

```
proy1.py > ...
1  from os import system
2  from lista_simplee import ListaGanador
3  from lista_simplee import ListaJugador
4  from lista_simplee import ListaRegalos
5  from MatrizDispersa import MatrizDispersa
6  |
7  > def leer_xml(ruta_xml, ruta2_xml): ...
153
154 > def generar_puzzle2(ruta_xml): ...
238
239 > def generar_puzzle(ruta_xml): ...
335
336 > def generar_paracomun(): ...
369
370 > def graphviz_todos(): ...
388
389 > def entrega_regalos(): ...
430
431 > def graphvizCola_completa(): ...
449
450 > def graphvizCola_media(): ...
468
469 > def graphviz_pila_completa(): ...
488
489 > def graphviz_pila_mitad(): ...
```


Lectura XML

- En este caso, la lectura se hizo a través de ElementTree.
- En las capturas se puede ver la estructura del archivo.

```
jug.xml
1  <?xml version="1.0" encoding="UTF-8"?>
2  <jugadores>
3    <jugador>
4      <datospersonales>
5        <nombre>Felix6</nombre>
6        <edad>20</edad>
7      </datospersonales>
8      <movimientos>10</movimientos>
9      <tamaño>5</tamaño>
10     <figura>Arbol de Navidad</figura>
11     <puzzle>
12       <celda f="1" c="0" />
13       <celda f="1" c="4" />
14       <celda f="1" c="3" />
15       <celda f="2" c="3" />
16       <celda f="3" c="4" />
17       <celda f="4" c="1" />
18       <celda f="4" c="2" />
19       <celda f="4" c="3" />
20       <celda f="4" c="4" />
21       <celda f="0" c="2" />
22     </puzzle>
23     <solucion>
24       <celda f="1" c="2" />
25       <celda f="2" c="1" />
26       <celda f="2" c="2" />
27       <celda f="2" c="3" />
28       <celda f="3" c="0" />
29       <celda f="3" c="1" />
30       <celda f="3" c="2" />
31       <celda f="3" c="3" />
32       <celda f="3" c="4" />
33       <celda f="4" c="2" />
```

```
jug.xml
1  <?xml version="1.0" encoding="UTF-8"?>
2  <jugadores>
3    <jugador> ...
35  </jugador>
36  <jugador> ...
68  </jugador>
69  <jugador> ...
101 </jugador>
102 <jugador> ...
134 </jugador>
135 <jugador> ...
167 </jugador>
168 <jugador> ...
200 </jugador>
201 <jugador> ...
233 </jugador>
234 <jugador> ...
266 </jugador>
267 <jugador> ...
299 </jugador>
300 <jugador> ...
332 </jugador>
333 <jugador> ...
365 </jugador>
366 <jugador> ...
398 </jugador>
399 </jugadores>
```

Lectura XML

- Se crearon listas de los datos y más adelante se procedía a recorrerlas para almacenar los datos del xml en las estructuras. En este caso primero se llenaba la estructura de jugadores y luego del cálculo de puntos se hacía el llenado a la estructura de ganadores, de igual forma con los regalos.

```
def leer_xml(ruta_xml, ruta2_xml):
    import xml.etree.ElementTree as ET
    try:
        #variable para cada jugador
        i = 1
        #variables para sumar puntos
        global puntos
        #lectura de ruta_xml (jugadores)
        ruta_xml = ruta_xml
        xml_file = open(ruta_xml, encoding="utf-8-sig")
        global jug
        global jug_comun
        jug = ListaGanador()
        #Esta va a guardarlos todos, pero se van a procesar ganadores y así con la otra lista
        #esta lista va a servir también para hacer búsquedas y eso
        jug_comun = ListaJugador()

        global jug_consultas
        jug_consultas = ListaJugador()
        if xml_file.readable():
            datos = ET.fromstring(xml_file.read())
            lista_jugadores = datos.findall("jugador")
            print("Total de jugadores:", len(lista_jugadores))
            for jugador in lista_jugadores:
                puntos = 0
                lista_datos = jugador.findall("datospersonales")
                lista_movimientos = jugador.findall("movimientos")
                lista_tamaño = jugador.findall("tamaño")
                lista_figura = jugador.findall("figura")
                lista_puzzle = jugador.findall("puzzle")
                lista_solucion = jugador.findall("solucion")
                for datos in lista_datos:
```


lista_simplee.py: Listas Enlazadas

- Archivo encargado de crear listas para los diferentes objetos: jugador común, jugador ganador y regalos o premios.

```
lista_simplee.py > ListaRegalos
1  from ganador import Ganador
2  from jugador import Jugador
3  from premio import Premio
4
5  > class ListaGanador(): ...
236
237 > class ListaRegalos():|...
342
343 > class ListaJugador(): ...
```

- Tienen métodos similares pero se emplean de diferentes formas. Se verán en la próxima diapositiva.

lista_simplee.py:

Listas Enlazadas

- Entre sus métodos están agregar de último o primero (sea cola o pila), eliminar (hay para eliminar sólo el primero, por ejemplo), recorrer, enviar a otra lista...

```
class ListaGanador():
> def __init__(self): ...
> def estaVacia(self): ...
> def agregarUltimo(self,nombre,puntos): ...
> #si viene de 1 a 10
> def agregarPrimero(self, nombre, puntos): ...
> def recorrido(self): ...
> def env_nombres(self): ...
> def env_medio_nombres(self): ...
> def eliminar_primero1(self): ...
> def eliminar_solo_uno(self): ...
> def eliminar_primero2(self): ...
> def eliminar_ultimo(self): ...
> def eliminar_ult(self): ...
> def ordenar_menor_mayor(self): ...
> def ordenar_mayor_menor(self): ...
> def ordenar_may_men(self): ...
> def eliminar_mitad(self): ...
```

lista_simplee.py:

Listas Enlazadas

- Entre sus métodos están agregar de último o primero (sea cola o pila), eliminar (hay para eliminar sólo el primero, por ejemplo), recorrer, enviar a otra lista...

```
class ListaRegalos():
> def __init__(self): ...
> def estaVacia(self): ...
> def eliminar_solo_uno(self): ...
>     #si viene de 10 a 1
> def agregarUltimo(self,regalo,lugar): ...
>     #si viene de 1 a 10
> def agregarPrimero(self, regalo, lugar): ...
> def recorrido(self): ...
> def eliminar_primero1(self): ...
> def eliminar_primero2(self): ...
> def env_nombres(self): ...
> def eliminar_ultimo(self): ...

class ListaJugador():
> def __init__(self): ...
> def estaVacia(self): ...
```

lista_simplee.py:

Listas Enlazadas

- Entre sus métodos están agregar de último o primero (sea cola o pila), eliminar (hay para eliminar sólo el primero, por ejemplo), recorrer, enviar a otra lista...

```
class ListaJugador():
> def __init__(self): ...
>
> def estaVacia(self): ...
>
> def agregarUltimo(self,nombre,puntos, figura, tamaño, movimientos): ...
> #si viene de 1 a 10
> def agregarPrimero(self, nombre, puntos, figura, tamaño, movimientos): ...
>
> def eliminar_solo_uno(self): ...
>
> def recorrido(self): ...
>
> def env_nombres(self): ...
>
> def env_primeros10(self): ...
>
> def env_medio_nombres(self): ...
>
> def eliminar_primero(self): ...
>
> def eliminar_mitad(self): ...
> #esta es la que funciona para el caso
> def ordenar_menor_mayor(self): ...
> #supongo que igual debería recorrer y luego ordenarlo
> def ordenar_men_may(self): ...
>
> def ordenar_mayor_menor(self): ...
>
> def ordenar_may_men(self): ...
>
> def buscar(self, nombre): ...
```

Objetos de las listas

Son los objetos que manejan las estructuras mencionadas anteriormente.

```
ganador.py > Ganador > __init__  
1 class Ganador():  
2     def __init__(self, nombre, puntos):  
3         self.nombre = nombre  
4         self.puntos = puntos  
5         self.siguiente = None
```

```
premio.py > Premio > __init__  
1 class Premio():  
2     def __init__(self, regalo, lugar):  
3         self.regalo = regalo  
4         self.lugar = lugar  
5         self.siguiente = None
```

```
jugador.py > Jugador > __init__  
1 class Jugador():  
2     def __init__(self, nombre, puntos, figura, tamaño, movimientos):  
3         self.nombre = nombre  
4         self.puntos = puntos  
5         self.figura = figura  
6         self.tamaño = tamaño  
7         self.movimientos = movimientos  
8         self.siguiente = None
```