TECHNICAL MANUAL

Diana Estefania Berducido Domingo

202000277

IPC2 – Laboratorio

Aux. Jackeline Benítez

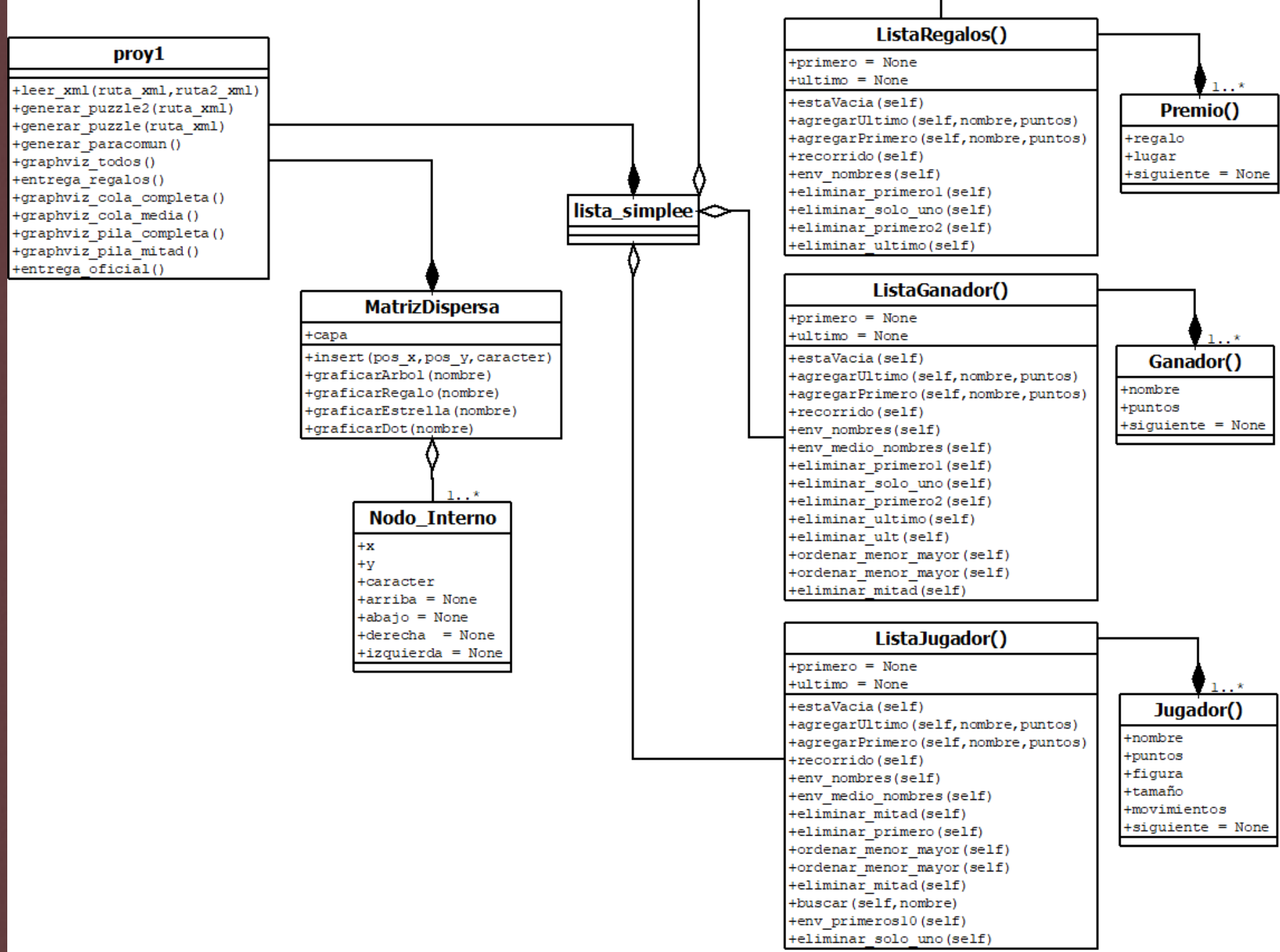
Programming used: OOP

- Through the implementation of data structures for simple linked lists and sparse arrays. These collect objects and data (players, winners, prizes, rows, columns) that it stores and mostly represents graphically.

Objective of the Program

- The goal is to obtain two files with XML structure.
- The XML is read and the structures of simple linked lists are filled to execute the queues and stacks.
- The user has several options: Generate a simulated gift delivery on console, Generate the puzzles of all players while they withdraw from their queue graphically reflected and Generate a simulated gift delivery graphically where the start of the queue and stack and half of the execution are observed.
- The general objective is to calculate, according to the input files, the scores of the players and know who would be the first 10 places. These first places are given a prize and that delivery is reflected as a queue and a pile. In addition to being able to observe the figure in its messy and orderly puzzles.

Class Diagram



Proy1 (MAIN)

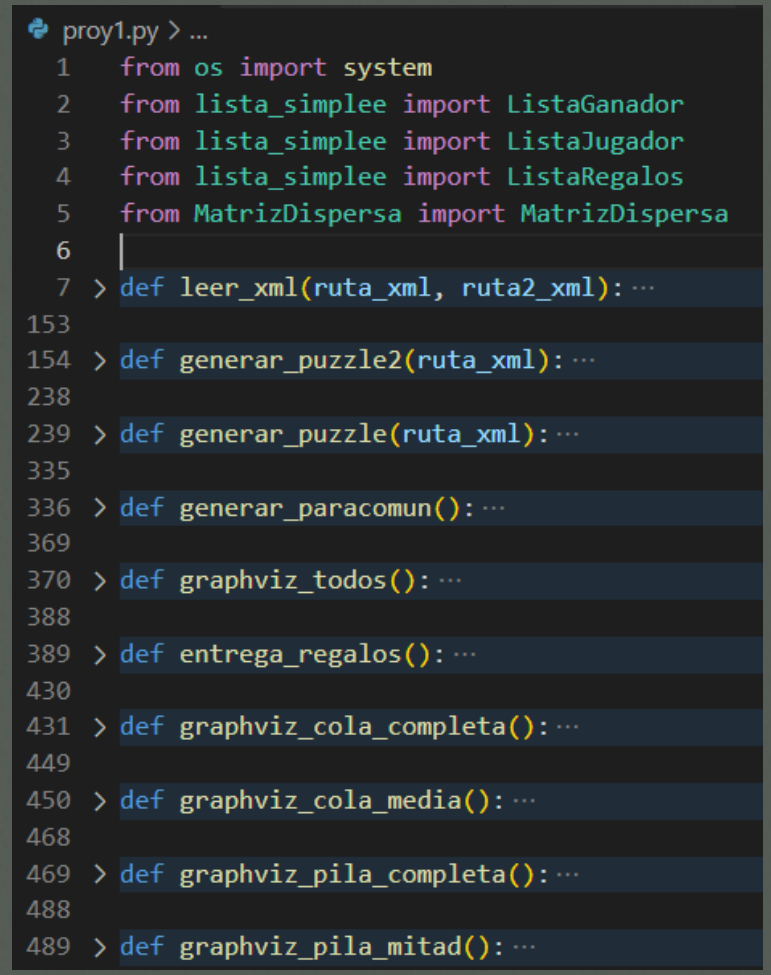
- After the declaration of several methods that help graph (next slides) has the start menu, where it asks for the input files and asks to select an action:

```
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)

- From here the other classes are called to execute their functions. The functions in this file are:

- And, all are responsible for the part of bringing data from the lists to graph them or display them in console.



XML reading

- In this case, the reading was done through ElementTree.
- In the screenshots you can see the structure of the file.

jug.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<jugadores>
    <jugador>
        <datospersonales>
            <nombre>Felix6</nombre>
            <edad>20</edad>
        </datospersonales>
        <movimientos>10</movimientos>
        <tamaño>5</tamaño>
        <figura>Arbol de Navidad</figura>
        <puzzle>
            <celda f="1" c="0" />
            <celda f="1" c="4" />
            <celda f="1" c="3" />
            <celda f="2" c="3" />
            <celda f="3" c="4" />
            <celda f="4" c="1" />
            <celda f="4" c="2" />
            <celda f="4" c="3" />
            <celda f="4" c="4" />
            <celda f="0" c="2" />
        </puzzle>
        <solucion>
            <celda f="1" c="2" />
            <celda f="2" c="1" />
            <celda f="2" c="2" />
            <celda f="2" c="3" />
            <celda f="3" c="0" />
            <celda f="3" c="1" />
            <celda f="3" c="2" />
            <celda f="3" c="3" />
            <celda f="3" c="4" />
            <celda f="4" c="2" />
```

jug.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<jugadores>
    <jugador>···
    </jugador>
    <jugador>···
    </jugador>
    <jugador>···
    </jugador>
    <jugador>···
    </jugador>
    <jugador>···
    </jugador>
    <jugador>···
    </jugador>
    <jugador>···
    </jugador>
    <jugador>···
    </jugador>
    <jugador>···
    </jugador>
    <jugador>···
    </jugador>
    <jugador>···
    </jugador>
    <jugador>···
    </jugador>
    <jugador>···
    </jugador>
</jugadores>
```

XML reading

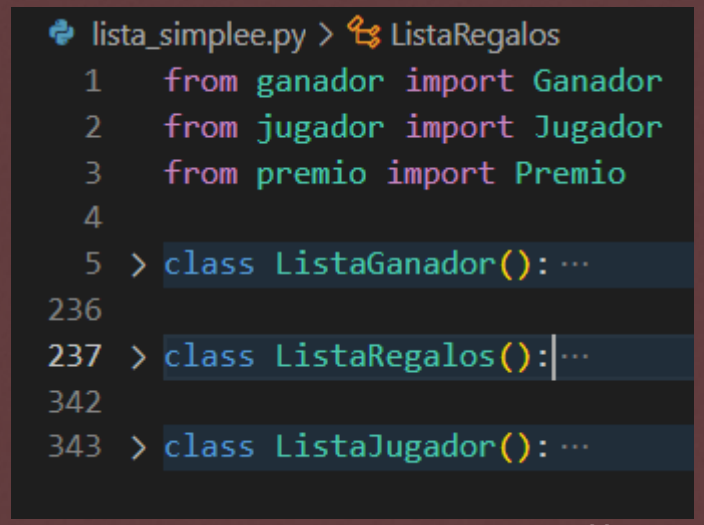
- The data was listed, and later traversed to store the xml data in the structures. In this case first the structure of players was filled and after the calculation of points the filling was made to the structure of winners, in the same way with the gifts.

```
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: Linked lists

- File in charge of creating lists for the different objects: common player, winning player and gifts or prizes.



- They have similar methods but are used in different ways. They will be seen on the next slide.

lista_simplee.py: Linked Lists

- Among its methods are add last or first (either queue or stack), delete (there is to delete only the first, for example), go through, send to another list ...

```
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: Linked Lists

- Among its methods are add last or first (either queue or stack), delete (there is to delete only the first, for example), go through, send to another list ...

```
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: Linked Lists

- Among its methods are add last or first (either queue or stack), delete (there is to delete only the first, for example), go through, send to another list ...

```
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):···
```

List objects

They are the objects that handle the structures mentioned above.

ganador.py > Ganador > __init__

```python
class Ganador():
    def __init__(self, nombre, puntos):
        self.nombre = nombre
        self.puntos = puntos
        self.siguiente = None
```

premio.py > Premio > __init__

```python
class Premio():
    def __init__(self, regalo, lugar):
        self.regalo = regalo
        self.lugar = lugar
        self.siguiente = None
```

jugador.py > Jugador > __init__

```python
class Jugador():
    def __init__(self, nombre, puntos, figura, tamaño, movimientos):
        self.nombre = nombre
        self.puntos = puntos
        self.figura = figura
        self.tamaño = tamaño
        self.movimientos = movimientos
        self.siguiente = None
```