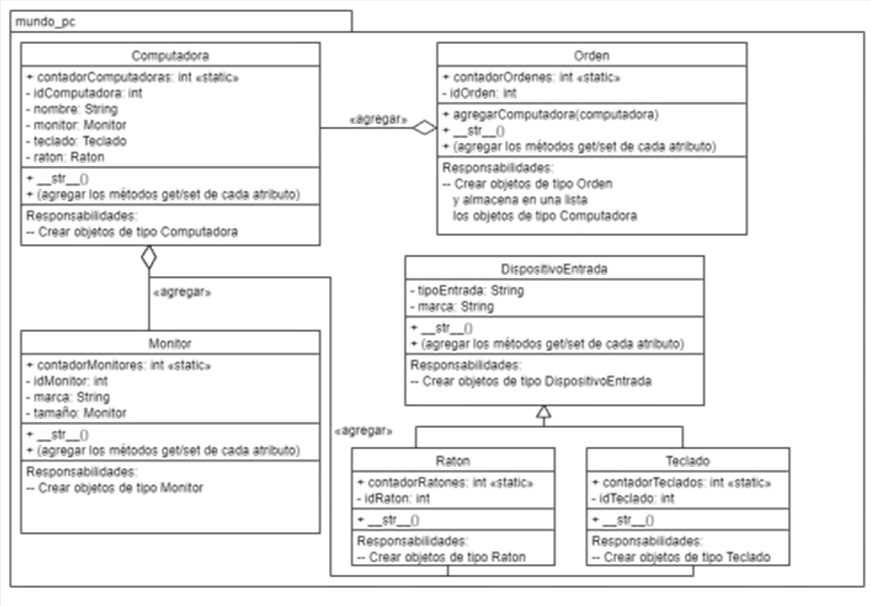
**LABORATORIO MUNDO PC**

Este es el diagrama de clases para la sección 18



Para comenzar con este proyecto, se recomienda iniciar con la creación de la clase que “no dependa” de otras para su creación. En este caso, comenzaremos con la clase Periférico, para luego heredar a la clase Ratón y Teclado.

Una vez resuelto, se creará la clase Monitor para así continuar con Computadora y por último, la clase Orden.

El ejercicio es muy similar a las prácticas realizadas anteriormente.

class Periferico:

    def \_\_init\_\_(self, tipo, marca) -> None:

        self.\_tipo = tipo

        self.\_marca = marca

    def \_\_str\_\_(self):

        return f'Periferico [Tipo: {self.\_tipo}, Marca: {self.\_marca}]'

Este es el código de la clase Periferico (además de los respectivos getters y setters). Como se ve, lo único que se modifica es su \_\_str\_\_()

Lo mismo ocurre para las clases Raton y Teclado que heredan desde Periferico.

class Teclado(Periferico):

    count\_keyboard = 0

    def \_\_init\_\_(self, tipo, marca) -> None:

*# uso el nombre de la clase para acceder a la variable de clase*

        Teclado.count\_keyboard += 1

        self.\_\_id\_teclado = Teclado.count\_keyboard

        super().\_\_init\_\_(tipo, marca)

    def \_\_str\_\_(self):

        return f'Teclado [id: {self.\_\_id\_teclado}] {super().\_\_str\_\_()}'

class Raton(Periferico):

    count\_mouse = 0

    def \_\_init\_\_(self, tipo, marca) -> None:

*# uso el nombre de la clase para acceder a la variable de clase*

        Raton.count\_mouse += 1

        self.\_\_id\_raton = Raton.count\_mouse

        super().\_\_init\_\_(tipo, marca)

    def \_\_str\_\_(self):

        return f'Raton [id: {self.\_\_id\_raton}] {super().\_\_str\_\_()}'

CABE RESALTAR QUE TODOS LOS ATRIBUTOS ID SE MANEJAN COMO PRIVADOS.

El código empieza a modificar cuando llegamos al objeto computadora.

Al momento de imprimir un objeto de tipo Computadora:

    def \_\_str\_\_(self) -> str:

        return f'''{self.\_nombre}, id:[{self.\_\_id\_pc}]

        {self.\_monitor}

        {self.\_teclado}

        {self.\_raton}

        '''

Tambien, al inicializar una instancia, sería bueno comprobar que el argumento que se recibe sea realmente del tipo que se necesita.

    def \_\_init\_\_(self, nombre, monitor, teclado, raton) -> None:

        if Computadora.validar\_monitor(monitor):

            self.\_monitor = monitor

        if Computadora.validar\_teclado(teclado):

            self.\_teclado = teclado

        if Computadora.validar\_raton(raton):

            self.\_raton = ratón

self.\_nombre = nombre

        self.\_\_id\_pc = Computadora.aumentar\_count\_pc()

EL caso de la clase Orden es similar, solo que implementa una lista de objetos de tipo Computadora como atributo.

    def \_\_init\_\_(self, \*computadoras) -> None:

        self.\_\_id\_orden = Order.aumentar\_count\_orden()

        self.\_computadoras = list()

        for e in computadoras:

            if(isinstance(e, Computadora)):

                self.\_computadoras.append(e)

            else:

                print(f'Error al intentar agregar {e} como elemento')

Para el caso, también es convencional hacer una comprobación de los objetos que se reciben.

Además esta clase contiene un método para agregar otro objeto Computadora a la Orden.

Para imprimir una instancia de tipo Orden uso este código:

    def print\_pcs(self):

        text\_pc = '\n'

        for e in self.\_computadoras:

            text\_pc += e.\_\_str\_\_() + '\n'

        else:

            return text\_pc

    def \_\_str\_\_(self):

        return f'''Orden {self.\_\_id\_orden}, computadoras:

{self.print\_pcs()}

        '''