Cátedra: Estructuras de Datos

Carrera: Licenciatura en Sistemas

Año: 2024 – 2do Cuatrimestre

Trabajo Práctico Nro 2:

P.O.O. en Python

Facultad de Ciencias de la Administración
Universidad Nacional de Entre Ríos

Objetivos

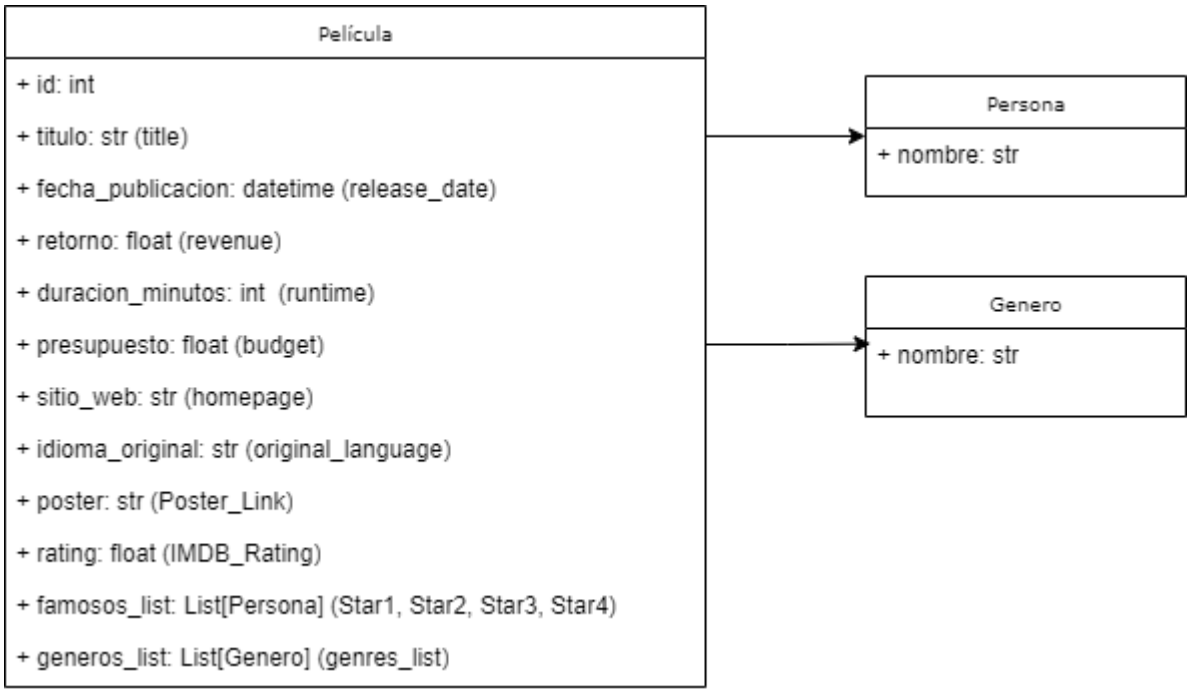
- El Trabajo Práctico N.º: 2 pretende que el alumno:
  - Aprenda a crear clases e instanciarlas.
  - Ponga en práctica los conceptos principales del paradigma orientado a objetos: abstracción, encapsulamiento, herencia y polimorfismo.
  - Comprenda como tratar con excepciones.

Condiciones de Entrega

- El trabajo práctico deberá ser:
  - Realizado en forma individual o en grupos de no más de tres alumnos.
  - Cargado en la sección del Campus Virtual correspondiente, junto con los archivos de código fuente que requieran las consignas. Cada uno de ellos en sus respectivos formatos pero comprimidos en un único archivo con formato zip, rar, tar.gz u otro. Si el trabajo práctico fue realizado en grupo, deberá especificarse en el nombre del archivo los apellidos de los integrantes del mismo.
  - Entregado el **Lunes 09 de Septiembre de 2024.**
- El práctico será evaluado:
  - Con nota numérica y conceptual (Excelente, Muy Bueno, Bueno, Regular y Desaprobado), teniendo en cuenta la exactitud y claridad de las respuestas, los aportes adicionales fuera de los objetivos de los enunciados y la presentación estética.
- Las soluciones del grupo deben ser de autoría propia. Aquellas que se detecten como idénticas entre diferentes grupos o producto de una I.A. serán clasificadas como **MAL** para todos los involucrados en esta situación que será comunicada en la devolución.
- Los ejercicios que exijan codificación se valorarán de acuerdo a su exactitud, prolijidad (identación y otras buenas prácticas).

Ejercicios de Programación

1. “El Mamboretá” es un sitio web de noticias y recomendaciones de cine y series que para aumentar su cantidad de visitas ha decidido importar un catálogo con información de películas. El equipo editores necesita además contar con algunas funcionalidades de búsqueda motivo por el cual se le encarga el desarrollo. La información a registrar es:



   a) Construir las clases que considere necesarias para soportar la información presentada.

   b) Para cada una de las clases del punto anterior defina un constructor parametrizado y el método **`__str__()`**.

   c) Identifique los campos clave de cada una de las clases y programe el método **`__eq__()`**.

   d) Programe la clase **`MamboretaAdmin`** como clase derivada de **`MamboretadminAbstract`** y programe los métodos abstractos en ella definida.

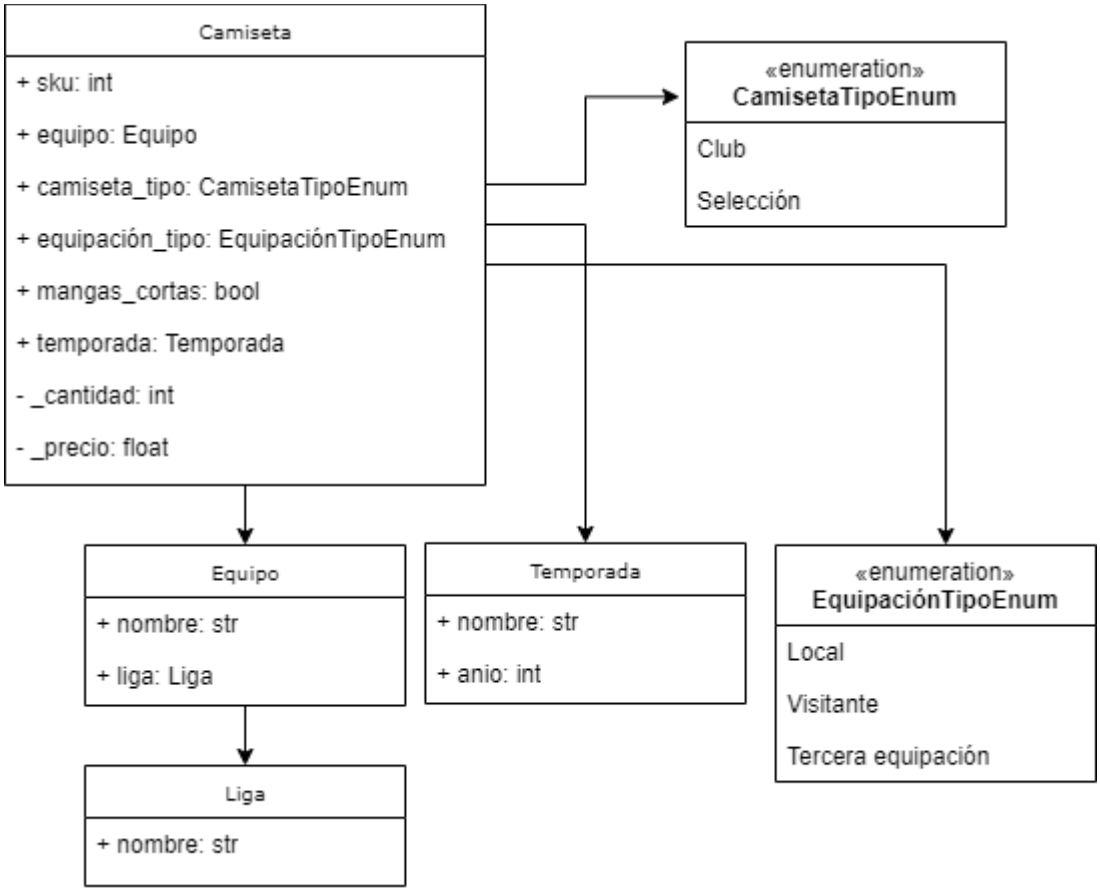
```
from abc import ABC, abstractmethod
from typing import List
from pelicula import Pelicula
from persona import Persona
from genero import Genero

class MamboretaAdminAbstract(ABC):
    """
    Clase abstracta que define la estructura básica para un administrador de películas.
```

```
    Permitiendo filtrar y organizar las mismas según diferentes criterios.
    """

    def __init__(self) -> None:
        """
        Inicializa una nueva instancia.
        """
        self.lista : List[Pelicula] = []

    @abstractmethod
    def procesar_archivo(self, ruta : str) -> None:
        """
        Añade todas las películas desde el archivo CSV cuya ruta se pasa como parámetro.

        Args:
            ruta (str): ruta hacia el archivo que se desea importar.
        """
        pass

    @abstractmethod
    def __str__(self) -> str:
        """
        Concatena todas los elementos de lista en un único str.

        Returns:
            str: Una cadena que representa el objeto.
        """
        pass

    @abstractmethod
    def filtrar_por_genero(self, genero: Genero) -> List[Pelicula]:
        """
        Filtra las películas en la lista según el género especificado.

        Args:
            genero (Genero): El género por el cual se filtrarán las películas.

        Returns:
            List[Pelicula]: Una lista de películas que pertenecen al género especificado.
        """
        pass

    @abstractmethod
    def filtrar_por_persona(self, persona: Persona) -> List[Pelicula]:
        """
        Filtra las películas en la lista según un actor específico

        Args:
            persona (Persona): La persona por la cual se filtrarán las películas.

        Returns:
            List[Pelicula]: Una lista de películas asociadas con la persona especificada.
        """
        pass

    @abstractmethod
    def filtrar_companieros(self, persona1: Persona, persona2: Persona) -> List[Pelicula]:
        """
        Filtra las películas en las que dos personas específicas han trabajado juntas.

        Args:
            persona1 (Persona): La primera persona.
            persona2 (Persona): La segunda persona.

        Returns:
```

```
            List[Pelicula]: Una lista de películas en las que ambas personas han colaborado.
        """
        pass

    @abstractmethod
    def top_n(self, n: int = 50) -> List[Pelicula]:
        """
        Devuelve en orden descendente de rating las 'n' películas más destacadas

        Args:
            n (int): El número de películas a devolver.

        Returns:
            List[Pelicula]: Una lista con las 'n' mejores películas.
        """
        pass

    @abstractmethod
    def fracasos_comerciales(self, umbral: float = 0.0) -> List[Pelicula]:
        """
        Filtra y devuelve las películas que han sido fracasos comerciales.
        Por defecto, el umbral es 0.0, lo que significa que se considerará fracaso
        cualquier película cuyo retorno sea menor que su presupuesto.

        Args:
            umbral (float, optional): El umbral para determinar el fracaso comercial.
                                      Si la diferencia entre retorno y presupuesto es menor
                                      que este valor, la película se considera un fracaso.
                                      El valor predeterminado es 0.0.

        Returns:
            List[Pelicula]: Una lista de películas que no tuvieron éxito comercial.
        """
        pass
```

e) Demuestre el correcto funcionamiento de los puntos anteriores.

2. **Castolo Kits** es una tienda en línea que se dedica a la comercialización de camisetas de fútbol que le encarga el desarrollo del catálogo de productos de su nuevo sitio. Los datos a registrar son los siguientes:



a) Construir las clases y/o enumeraciones que considere necesarias para soportar la información presentada.

b) Para cada una de las clases del punto anterior defina un constructor parametrizado y el método **`__str__()`**.

c) Identifique los campos clave de cada una de las clases y programe el método **`__eq__()`**.

d) Defina la propiedad solo lectura agotado que indique cuando no quedan más productos del tipo en stock.

e) Para todas las operaciones que configuren un valor para el atributo precio de la clase Camiseta establezca controles que prevengan la carga de valores inválidos y arrojen la excepción apropiada.

f) Programe la clase **`CastoloAdmin`** como clase derivada de **`CastoloAdminAbstract`** y programe los métodos abstractos en ella definida.

```
from abc import ABC, abstractmethod
from typing import List, Dict

from camiseta import Camiseta
from equipacion_tipo_enum import EquipacionTipoEnum
from liga import Liga

class CastoloAdminAbstract(ABC):

	def __init__(self) -> None:
		"""Inicializa la clase con una lista vacía de camisetas."""
		self.lista: List[Camiseta] = []

	def append(self, camiseta: Camiseta) -> None:
		"""Agrega una camiseta a la lista de camisetas.

		Args:
			camiseta (Camiseta): La camiseta que se desea agregar a la lista.
		"""
		self.lista.append(camiseta)

	def __str__(self) -> str:
		"""Concatena en un único str todas la camisetas

		Returns:
			str: Una cadena con las camisetas, separadas por líneas.
		"""
		return "\n".join(str(camiseta) for camiseta in self.lista)

	@abstractmethod
	def filtrar_por_liga(self, liga: Liga) -> List[Camiseta]:
		"""Filtra las camisetas por liga.

		Args:
			liga (Liga): La liga por la cual se desea filtrar.

		Returns:
			List[Camiseta]: Una lista de camisetas que pertenecen a la liga especificada.
		"""
		pass

	@abstractmethod
	def filtrar_por_equipacion_tipo(self, equipacion_tipo: EquipacionTipoEnum) ->
List[Camiseta]:
		"""Filtra las camisetas por tipo de equipación.

		Args:
			equipacion_tipo (EquipacionTipoEnum): El tipo de equipación por el cual se desea
filtrar.

		Returns:
			List[Camiseta]: Una lista de camisetas que corresponden al tipo de equipación
especificado.
		"""
		pass

	@abstractmethod
	def filtrar_temporada_actual(self, solo_clubes: bool) -> List[Camiseta]:
		"""Filtra las camisetas que pertenecen a la temporada actual.

		Args:
```

```
                solo_clubes (bool): Si es True, solo filtra camisetas de clubes; si es False,
incluye selecciones nacionales.

            Returns:
                List[Camiseta]: Una lista de camisetas que pertenecen a la temporada actual.
            """
            pass

      @abstractmethod
      def stock_agotado(self) -> List[Camiseta]:
            """Devuelve una lista de camisetas que están agotadas en stock.

            Returns:
                List[Camiseta]: Una lista de camisetas con stock agotado.
            """
            pass

      @abstractmethod
      def mas_costosa(self) -> Camiseta:
            """Devuelve la camiseta más costosa de la lista.

            Returns:
                Camiseta: La camiseta más costosa.
            """
            pass

      @abstractmethod
      def totales_por_liga(self) -> Dict[Liga, int]:
            """Devuelve un diccionario con el número total de camisetas por liga.

            Returns:
                Dict[Liga, int]: Un diccionario donde las claves son ligas y los valores el
número total de camisetas de cada liga.
            """
            pass
```

g) Demuestre el correcto funcionamiento de los puntos anteriores.

Opcionales

3. ARSAT ha suministrado a la Ministerio de Economía los datos de los puntos Wi-Fi con acceso libre en la región. Se le encarga como tarea definir las estructuras de datos que soportarán en memoria la información y una serie de funciones de consulta.

   a) Construir las clases que considere necesarias para soportar la información presentada.

   b) Para cada una de las clases del punto anterior defina un constructor parametrizado y el método **`__str__()`**.

   c) Identifique los campos clave de cada una de las clases y programe el método **`__eq__()`**.

   d) Programe las siguientes funciones:

1. `def puntos_por_municipio(lista: List[PuntoWiFi], municipio: Municipio) → List[PuntoWIFI]`: Devuelve los puntos Wi-Fi por municipio
2. `def ordenar(lista: List[PuntoWiFi] → None`: Ordena los puntos Wi-Fi por nombre de provincia, luego por departamento y por último por municipio.
3. `def totales_por_provincia(lista: List[PuntoWiFi] → Dict[Provincia, int]`: Totaliza la cantidad de puntos Wi-Fi por provincia.

e) Demuestre el correcto funcionamiento de los puntos anteriores.

4. Construir una interfaz gráfica para el ejercicio N.º 1 que muestre los posters de las películas descargándolas desde su URL.