Cátedra: Estructuras de Datos

Carrera: Licenciatura en Sistemas

Año: 2022 – 1er 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 sus propias clases y crear instancias de las mismas.
  - 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.
  - Ser 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.
  - Ser entregado:
    - El trabajo práctico definitivo el **Viernes 16 de Septiembre de 2022.**
- 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 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).

Clases y Objetos

1. La Biblioteca de la Facultad de Ciencias de la Administración necesita una aplicación para registrar y consultar los **Libros** con los que cuenta. Los datos a representar son:

Libro	
Atributo	Tipo de Datos
Número de Inventario	int
Título	str
Autor	Autor
Categoría	Categoría
Año de Publicación	int

Autor	
Atributo	Tipo de Datos
Apellido	str
Nombre	str

Categoría (Área de Conocimiento)	
Atributo	Tipo de Datos
Nombre	str

a) Programe las clases que considera necesarias para dar solución al problema planteado.

b) Modifique las clases creadas en el punto anterior agregando:
   i. Un constructor que reciba por parámetro valores para inicializar cada una de las propiedades o atributos.
   ii. El método especial **`__str__()`** que transforme en un único string los valores que configuran el estado actual del objeto.
   iii. El método especial **`__eq__()`** que indique cuando dos objetos de la misma clase son iguales.

c) Cree un archivo con nombre **`libros_cliente.py`** y dentro de él:
   i. Programe las siguientes funciones:
      1. **`imprimir(libros : list)`** → **`None`** que imprima los datos de la lista pasada por parámetro.
      2. **`filtrar_por_categoria(libros : list, cat : Categoria)`** → **`list`** que retorne todos los libros cuya categoría coincide con la pasada por parámetro.
      3. **`filtrar_por_autor(libros : list, autor : Autor)`** → **`list`** que devuelva los libros cuyo autor coincide con el pasado por parámetro.
      4. **`filtrar_por_anio(libros : list, anio : int)`** → **`list`** que devuelva los

libros cuyo año de publicación coincide con el pasado por parámetro.

ii. En la sección principal del programa:
   1. Cree al menos 3 instancias de **`Libro`**.
   2. Ponga a prueba **TODAS** las funciones programadas en el punto anterior.

2. La Veterinaria Bichitos le encarga el desarrollo de un software que le permita llevar registro de sus pacientes. Los datos a registrar son:

Mascota	
Atributo	Tipo de Datos
Número de Registro	int
Nombre	str
Raza	Raza
Año de Nacimiento	int
Dueño/Amo/Responsable	Persona

Raza	
Atributo	Tipo de Datos
Nombre	str
Especie	Especie

Especie	
Atributo	Tipo de Datos
Nombre	str

Persona	
Atributo	Tipo de Datos
Apellido	str
Nombre	str
Documento	str

a) Construya la/s clase/s que considere necesarias para dar solución al problema.

b) Para cada una de las clases programe:
   i. Un constructor que reciba por parámetro valores para inicializar cada una de las propiedades o atributos.
   ii. Los métodos especiales: **`__str_(), __repr__() y __eq__()`**.

c) Defina la para la clase **`Mascota`** la propiedad calculada **`edad.`** Esta propiedad deberá calcularse a partir como la diferencia entre el año actual y el año de nacimiento de la **`Mascota`**. La misma deberá ser no modificable.

d) Cree un archivo con nombre **`bichitos_cliente.py`** y dentro de él programe las

siguiente funciones de consulta:

i. **`imprimir(mascotas : list) → None:`** que imprima por pantalla los datos contenidos en el array pasado por parámetro.
ii. **`filtrar_gerontes(mascotas : list) → list:`** devuelve las mascotas con 13 o más años de edad.
iii. **`filtrar_por_especie(mascotas : list, especie : Especie):`** devuelve las mascotas cuya especie coincide con `especie`.
iv. **`max_mascotero(mascotas : list) → Persona:`** devuelve la persona que tiene más mascotas a su cargo.

e) En la sección principal del archivo **`bichitos_cliente.py`**:
   i. Cree al menos 5 instancias de **`Mascota`**.
   ii. Almacene las referencias de tipo **`Mascota`** en una lista.
   iii. Muestre los datos de las mismas por pantalla.
   iv. Muestre el resultado de invocar las funciones programadas en el punto anterior.

Herencia, Clases Abstractas y Excepciones

3. La empresa “Floripa!” que se dedica a la comercialización de indumentaria y accesorios deportivos desea implementar un sistema de control de acceso de sus empleados. La información a registrar es:

Empleado	
Atributo	Tipo de Datos
Legajo	int
Documento	str
Apellido	str
Nombre	str
Oficina	Oficina

Oficina	
Atributo	Tipo de Datos
Nombre	str
HoraEntrada	datetime.time
HoraSalida	datetime.time

Marcación	
Atributo	Tipo de Datos
Número de Registro	int
Empleado	Empleado
FechaHora	datetime.datetime
Tipo	MarcacionTipo

MarcacionTipo
Valores posibles: Entrada, Salida

a) Programe las clases que considera necesarias para dar solución al problema planteado.

b) Modifique las clases creadas en el punto anterior agregando:
   i. Un constructor que reciba por parámetro valores para inicializar cada una de las propiedades o atributos.
   ii. Los métodos especiales: **`__str_(), __repr__() y __eq__()`**.

c) Modifique la clase **`Empleado`** implementando el método especial **`__lt__(self, other)`** de forma que indique que una instancia de Empleado es menor que otra comparando los atributos legajo.

d) Modifique la clase **`Marcacion`** de forma que:
   i. Cuente con una propiedad de clase que almacene el último **número de registro** asignado.
   ii. Modifique el constructor de forma que a cada nueva instancia se asigne un **número de registro** y posteriormente se incremente el atributo de clase.
   iii. Defina un método de acceso solo lectura que encapsule el atributo **número de registro**. En caso que quiera modificarse el valor del atributo arroje una excepción.

e) Construya la clase **`MarcacionesAdmin`** como clase que extienda a **`MarcacionesAdminAbstract`** e implemente sus métodos abstractos:

```python
from abc import ABC, abstractmethod
from empleado import Empleado
from marcacion import Marcacion

class MarcacionesAdminAbstract(ABC):

    def __init__(self) -> None:
        self.marcaciones = list()

    def __len__(self) -> int:
        """Indica la cantidad de marcaciones almacenadas.
        Returns:
            int: cantidad de elementos almacenados actualmente en marcaciones.
        """
        return len(self.marcaciones)

    def __getitem__(self, key: int) -> Marcacion:
        """Obtiene el elemento en la posición indicada por key."""
        return self.marcaciones[key]

    def __delitem__(self, key: int) -> None:
        """Elimina la marcación ubicada en la posición indicada por key."""
        del self.marcaciones[key]

    def __str__(self) -> str:
        res = ""
        for elem in self.marcaciones:
            res += "{elem}\n".format(elem=str(elem))
        return res
```

```python
    @abstractmethod
    def agregar(self, marcacion : Marcacion) -> None:
        """Agrega la marcación pasada por parámetro al final de la lista de marcaciones
        Args:
            marcacion (Marcacion): instancia de clase marcación que se
            va a agregar al final de la lista de marcaciones.
        """
        pass

    @abstractmethod
    def empleados(self) -> list:
        """Devuelve todos los empleados de los que se tiene registro de marcación(no repetir
resultados).
        Returns:
            list: lista formada por las ocurrencias únicas de
            empleados dentro de la lista de marcaciones
        """
        pass

    @abstractmethod
    def filtrar_por_empleado(self, empleado: Empleado) -> list:
        """Devuelve todas las marcaciones de un empleado."""
        pass

    @abstractmethod
    def filtrar_por_tipo(self, tipo: Marcacion) -> list:
        """Devuelve todas las marcaciones del tipo especificado por parámetro."""

    @abstractmethod
    def llegadas_tarde(self) -> list:
        """Devuelve las marcaciones realizadas fuera del horario de ingreso."""

    @abstractmethod
    def ordenar_legajo(self) -> None:
        """Ordena las marcaciones por legajo de empleado y luego por fecha/hora."""
        pass

    @abstractmethod
    def ordenar_apellido_nombre(self) -> None:
        """Ordena las marcaciones por apellido y nombre del empleado, luego por fecha/hora."""
        pass
```

4. Programe el archivo **`floripa_cliente.py`** como archivo principal de la aplicación. En él realice las siguientes tareas:
   a) Crear al menos 5 instancias de **`Empleado`**.
   b) Crear al menos 15 instancias de **`Marcacion`**.
   c) Invocar a **TODOS** los métodos de la clase **`MarcacionesAdmin`** y demostrar su correcto funcionamiento.

5. **`CD Market`** es una empresa que se dedica a la comercialización de videojuegos que ha decidido contratarlo para programar su sitio Web. En el mismo se deberá exhibir el catálogo de títulos con los que cuenta. La información a registrar es la siguiente:

Videojuego	
Atributo	Tipo de Datos
Titulo	str
Género	Genero
Plataformas	list

Descripción	str
Precio	float
Empresa Desarrolladora	Empresa
Empresa Distribuidora	Empresa
Fecha de Lanzamiento	datetime.date
Ranking Metacritic	float

Género	
Atributo	Tipo de Datos
Valores posibles: *Plataformas, Lucha, Acción/Aventuras, FPS, RPG, Deportes, Carreras, Aventura Animada, Otro.*	

Plataforma	
Atributo	Tipo de Datos
Nombre	str
Es Portátil	bool

Empresa	
Atributo	Tipo de Datos
Nombre	str

a) Construya las clases necesarias para dar mantener la información solicitada. Las mismas deberán contar con:
   i. Constructor con parámetros.
   ii. Los métodos especiales: **`__str_(), __repr__() y __eq__()`**.

b) Modifique Videojuego de forma que no acepte para la propiedad Ranking Metacritic ni asignación de valores superiores a 10 ni menores que 0. Arroje una excepción cuando esto suceda.

c) Programe la clase **`VideojuegosAdmin`** como clase derivada de la clase abstracta **`VideojuegoAdminAbstract`** implementando todos sus métodos abstractos.

```
from abc import ABC, abstractmethod
from plataforma import Plataforma
from genero import Genero
from empresa import Empresa
from videojuego import Videojuego

class VideojuegosAdminAbstract(ABC):

    def __init__(self) -> None:
        self.videojuegos = []

    def __len__(self) -> int:
        """Indica la cantidad de videojuegos registrados.
        Returns:
```

```
            int: cantidad de elementos almacenados actualmente en videojuegos.
        """
        return len(self.videojuegos)

    def __getitem__(self, key: int) -> Videojuego:
        """Obtiene el elemento en la posición indicada por key."""
        return self.videojuegos[key]

    def __delitem__(self, key: int) -> None:
        """Elimina el videojuego ubicado en la posición indicada por key."""
        del self.videojuegos[key]

    @abstractmethod
    def __str__(self) -> str:
        """Concatena en un str todos los videojuegos del catálogo."""
        pass

    @abstractmethod
    def agregar(self, videojuego : Videojuego) -> None:
        """Agrega el parámetro al final de videojuegos
        Args:
            videojuego (Videojuego): instancia de clase videojuego que se
            va a agregar al final de la lista de videojuegos.
        """
        pass

    @abstractmethod
    def filtrar_por_desarrolladora(self, desarrolladora: Empresa) -> list:
        """Devuelve los videojuegos desarrollados por la empresa pasada por parámetro."""
        pass

    @abstractmethod
    def filtrar_por_distribuidora(self, distribuidora: Empresa) -> list:
        """Devuelve los videojuegos distribuídos por la empresa pasada por parámetro."""
        pass

    @abstractmethod
    def filtrar_por_genero(self, genero: Genero) -> list:
        """Devuelve los videojuegos del género pasado por parámetro. """
        pass

    @abstractmethod
    def cantidad_por_plataforma(self) -> list:
        """Indica la cantidad de videojuegos por plataforma. """
        pass

    @abstractmethod
    def ordenar_titulo(self) -> None:
        """Ordena los videojuegos por titulo."""
        pass

    @abstractmethod
    def ordenar_mejores_primero(self) -> None:
        """Ordena los videojuegos por ranking descendente. """
        pass
```

d) Programe el archivo **cdmarket_cliente.py** como principal y allí pruebe el funcionamiento de las clases y métodos programadas en los puntos anteriores.