+++ date = '2025-07-22' draft = false title = 'Practica 3: Paradigma Orientado a Objetos' +++

Introducción

Desarrollo

memory_management.py

Analizando el codigo de *memory_managent.py* podemos observar que se crea una clase para llevar un control de la memoria.

```
class MemoryManagement:
    '''Class to manage memory usage'''
    def __init__(self):
        self.heap_allocations = 0
        self.heap_deallocations = 0

    def increment_heap_allocations(self, size):
        '''Increment heap allocations'''
        self.heap_allocations += size

    def increment_heap_deallocations(self, size):
        '''Increment heap deallocations '''
        self.heap_deallocations += size

    def display_memory_usage(self):
        '''Display memory usage'''
        print(f"Heap allocations: {self.heap_allocations} bytes")
        print(f"Heap deallocations: {self.heap_deallocations} bytes")

memory_management = MemoryManagement()
```

Dentro del codigo de la clase primero tenermos un constructor con la función *init(self)* que se recibe como parametro a si mismo para inicializar sus dos atributos a cero.

```
def __init__(self):
    self.heap_allocations = 0
    self.heap_deallocations = 0
```

Una vez declarado su constructor pasamos a los metodos, en particular esta clase cuenta con tres metodos el primer metodo sive para aumentar el contador de asignación de memoria en el heap, el segundo para aumentar el contador de desasignación de memoria en el heap y su ultimo metodo es para mostrar la cantidad de bytes que se asignaron y desasignaron.

```
    def increment_heap_allocations(self, size):
        '''Increment heap allocations'''
        self.heap_allocations += size

    def increment_heap_deallocations(self, size):
        '''Increment heap deallocations '''
        self.heap_deallocations += size

    def display_memory_usage(self):
        '''Display memory usage'''
        print(f"Heap allocations: {self.heap_allocations} bytes")
        print(f"Heap deallocations: {self.heap_deallocations} bytes")
```

Finalmente se crea un objeto de clase **MemoryManagement** mediante su contructor, ya que este sera utilizado en el siguiente codigo.

```
memory_management = MemoryManagement()
```

biblioteca.py

En este codigo primero se hacen dos importaciones

```
import json
from memory_management import memory_management
```

json se utiliza para cargar y guardar los datos de los libros y los usuarios sentro de un archivo json y **memory_management** se importa para hacer uso del objeto que se creo en el codigo de *memory_management*.

Seguido de esto estan las declaraciones de clases la primera de ellas es una clase para el genero de los libros, esta clase intenta simular un **enum**

```
class Genre:
    '''Clase para definir los generos de los libros'''
    FICTION = "Ficcion"
    NON_FICTION = "No Ficcion"
    SCIENCE = "Ciencia"
    HISTORY = "Historia"
    FANTASY = "Fantasia"
    BIOGRAPHY = "Biografia"
    OTHER = "Otro"

    @classmethod
    def all_genres(cls):
        return [cls.FICTION, cls.NON_FICTION, cls.SCIENCE, cls.HISTORY,
cls.FANTASY, cls.BIOGRAPHY, cls.OTHER]
```

Despues tenemos la clase libro con su constructor, destructor y 2 metodos

```
class Book:
    '''Clase para definir los libros de la biblioteca'''
    def __init__(self, book_id, title, author, publication_year, genre, quantity):
        self.id = book_id
        self.title = title
        self.author = author
        self.publication_year = publication_year
        self.genre = genre
        self.quantity = quantity
        memory_management.increment_heap_allocations(1)

    def __del__(self):
        memory_management.increment_heap_deallocations(1)

    def to_dict(self):
        '''Método para convertir los datos del libro en un diccionario'''
        return {
            "id": self.id,
            "title": self.title,
            "author": self.author,
            "publication_year": self.publication_year,
            "genre": self.genre,
            "quantity": self.quantity
        }

    @staticmethod
    def from_dict(data):
        '''Método para crear un objeto libro a partir de un diccionario'''
        return Book(
            data["id"],
            data["title"],
            data["author"],
            data["publication_year"],
            data["genre"],
            data["quantity"]
        )
```

Dentro del contrustor y del destructor podemos observar que se utiliza el objeto **memory_management** que se importo al principio del programa, cuando se crea un libro se incrementa la memoria en el heap y cuando se destruye incrementa el contador de desasignaciones. Ahora pasamos a la clase de **DigitalBook** que hereda de la clase **Book**, esto le podra permitir utilizar los mismos metodos de esta clase y además se puede aplicar polimorfismo.

```
class DigitalBook(Book):
    '''Clase para definir los libros digitales de la biblioteca'''
    def __init__(self, book_id, title, author, publication_year, genre, quantity,
```

```
file_format):
        '''Constructor de la clase DigitalBook'''
        super().__init__(book_id, title, author, publication_year, genre,
quantity)
        self.file_format = file_format

    def to_dict(self):
        '''Método para convertir los datos del libro digital en un diccionario'''
        data = super().to_dict()
        data["file_format"] = self.file_format
        return data

    @staticmethod
    def from_dict(data):
        '''Método para crear un objeto libro digital a partir de un diccionario'''
        return DigitalBook(
            data["id"],
            data["title"],
            data["author"],
            data["publication_year"],
            data["genre"],
            data["quantity"],
            data["file_format"]
        )
```

En esta clase podemos ver que en el contructor se manda a llamar el constructor de la clase superior **Book**, dentro de los metodos de la clase podemos ver aplicado el polimorfismo, estos metodos hacen lo mismo pero de fomra diferente a los de clase superior ya en esta clase tenemos un atributo adicional.

En la clase **Member** es muy parecida a lo que vemos en la clase **Book**

```
class Member:
    '''Clase para definir los miembros de la biblioteca'''
    def __init__(self, member_id, name):
        '''Constructor de la clase Member'''
        self.id = member_id
        self.name = name
        self.issued_books = []
        memory_management.increment_heap_allocations(1)

    def __del__(self):
        memory_management.increment_heap_deallocations(1)

    def to_dict(self):
        '''Método para convertir los datos del miembro en un diccionario'''
        return {
            "id": self.id,
            "name": self.name,
            "issued_books": self.issued_books
        }
```

```
    @staticmethod
    def from_dict(data):
        '''Método para crear un objeto miembro a partir de un diccionario'''
        member = Member(data["id"], data["name"])
        member.issued_books = data["issued_books"]
        return member
```

La ultima clase que tenemos en este codigo es **Library** que su objetivo es llevar el control de los libros y los miembros por ello sus atributos son una lista de libros y otra de miembros.

```
class Library:
    '''Clase para definir la biblioteca'''
    def __init__(self):
        '''Constructor de la clase Library'''
        self.books = []
        self.members = []
        memory_management.increment_heap_allocations(1)
```

En esta clase también estan todos los metodos para realizar acciones de la biblioteca, como buscar libros o miembros, prestar libros, regresar libros, etc.

Por último tenemos la función main que es la función principal del programa donde tenemos un menu para que el usuario pueda realizar la acción que desee.

```
def main():
   '''Función principal para ejecutar el programa'''
   library = Library()
   library.load_library_from_file("library.json")
   library.load_members_from_file("members.json")

   while True:
       print("\nMenu de sistema de manejo de biblioteca\n")
       print("\t1. Agregar un libro")
       print("\t2. Mostrar libros disponibles")
       print("\t3. Agregar un miembro")
       print("\t4. Prestar libro")
       print("\t5. Devolver libro")
       print("\t6. Mostrar miembros disponibles")
       print("\t7. Buscar miembro")
       print("\t8. Guardar y salir")
       choice = int(input("Indica tu opcion: "))

       if choice == 1:
           book_id = int(input("Ingresa ID del libro: "))
           title = input("Ingresa titulo del libro: ")
           author = input("Ingresa nombre del autor: ")
           publication_year = int(input("Ingresa el ano de publicacion: "))
           genre = input("Ingresa el genero del libro: ")
           quantity = int(input("Ingresa la cantidad de libros: "))
```

```
            is_digital = input("Es un libro digital? (s/n): ").lower() == 's'
            if is_digital:
                file_format = input("Ingresa el formato del archivo: ")
                book = DigitalBook(book_id, title, author, publication_year, genre,
quantity, file_format)
            else:
                book = Book(book_id, title, author, publication_year, genre,
quantity)
            library.add_book(book)
        elif choice == 2:
            library.display_books()
        elif choice == 3:
            member_id = int(input("Ingresa el ID del miembro: "))
            name = input("Ingresa el nombre del miembro: ")
            member = Member(member_id, name)
            library.add_member(member)
        elif choice == 4:
            member_id = int(input("Ingresa el ID del miembro: "))
            book_id = int(input("Ingresa el ID del libro: "))
            library.issue_book(book_id, member_id)
        elif choice == 5:
            member_id = int(input("Ingresa el ID del miembro: "))
            book_id = int(input("Ingresa el ID del libro: "))
            library.return_book(book_id, member_id)
        elif choice == 6:
            library.display_members()
        elif choice == 7:
            member_id = int(input("Ingresa el ID del miembro: "))
            library.search_member(member_id)
        elif choice == 8:
            library.save_library_to_file("library.json")
            library.save_members_to_file("members.json")
            print("Saliendo del programa\n")
            break
        else:
            print("Esta no es una opcion valida!!!\n")
```

biblioteca_web.py

En este ultimo codigo se hace uso del framework flask para implementar el programa en una pagina web, donde utilizan metodos de la clase Flask y también se crean funciones para realizar las mimsmas acciones que se realizan en el programa **biblioteca.py** dentro de la funcion **main()**.

Conclusión