Práctica 6

Objetivos:

Practicar conceptos de herencia y sobrecarga de operadores para modelar un sistema de registro diario que involucra empleados y clientes de una empresa. Se implementarán clases derivadas y se aplicará el concepto de polimorfismo.

Aplicación:

El personal de recepción de una empresa hace un registro diario de los empleados que acuden a trabajar (objetos *Empleado*) y de los clientes que visitan la empresa (objetos *Cliente*). Este registro diario va a estar modelado por una clase *RegistroDiario*.
La clase *Empleado*, hereda de la clase base *Ficha*. A continuación, se describen los atributos y métodos de cada una de las clases a implementar.

Clase Base: Ficha

Atributos (decida cómo deben declararse: privados o protegidos):
- `nombre`: nombre de la persona (cadena de texto).
- `edad`: edad de la persona (entero).
- `nacio`: hora de nacimiento, importando la clase *Time* de la práctica 3.

Métodos a implementar:
- `__init__`: inicializará sus atributos y tendrá como argumentos por defecto una cadena vacía ("") para el nombre, 0 para la edad y 12:00:00 AM para la hora.
- Utilice decoradores para el acceso con el mismo nombre de los atributos.
- `Visualizar()`: permitirá visualizar los datos del objeto y se redefinirá en las clases derivadas.

Clase Derivada: Empleado

Atributos adicionales (decida cómo deben declararse: privados o protegidos):
- `categoria`: categoría del empleado (ej. "Administrativo", "Técnico")(cadena de texto).
- `antiguedad`: años de antigüedad en la empresa (entero).

Métodos:
- `__init__`: inicializa nombre, edad, nacio, categoria y antiguedad.
- Utilice decoradores para el acceso con el mismo nombre de los atributos.
- `Visualizar()`: muestra los datos del empleado.

Clase Derivada: Cliente

Atributos adicionales (decida cómo deben declararse: privados o protegidos):
- dni: DNI del cliente (cadena de texto).

Métodos:
- `__init__`: inicializa nombre, edad, nacio, y dni.
- Utilice decoradores para el acceso con el mismo nombre del atributo.
- `Visualizar()`: muestra los datos del cliente.
- Implemente el método mágico `__eq__` (==) que permita comparar 2 clientes basándose en nombre y edad.

Clase RegistroDiario

Atributos (decida cómo deben declararse: privados o protegidos):
- `personas`: lista de instancias de *Ficha* (que pueden ser *Empleado* o *Cliente*).

Métodos:
- `agregar_persona(persona)`: agrega una persona al registro. Solo acepta objetos de tipo *Empleado* o *Cliente*.
- `visualizar_registro`(): llama al método visualizar de cada persona en el registro.
- `visualizar_empleados`(): muestra solo los empleados en el registro.
- `es_empleado(persona)`: devuelve True si la persona es un empleado, False en caso contrario.
- Implemente el método mágico `__getitem__` ([]) para acceder a los elementos en la lista de personas por índice.
- Implemente el método mágico `__add__` (+) para combinar dos registros en uno solo.

El programa principal se escribirá en un fichero `main.py` y contendrá un menú como el siguiente:

```
1. Introducir empleado
2. Introducir cliente
3. Buscar por nombre (y edad)
4. Mostrar registro diario
5. Mostrar empleados
6. Visualizar persona por índice
7. Combinar registros diarios
8. Salir
```

La opción 1 añadirá un empleado al objeto `registro`. La opción 2 añadirá un cliente al objeto `registro`.
La opción 3 solicitará el nombre de una persona y la edad y la buscará en el `registro`. Si encuentra a la persona, mostrará sus datos indicando si se trata de un empleado o de un cliente.

La opción 4 mostrará todas las personas registradas, tanto empleados como clientes, listando sus datos. La opción 5 mostrará únicamente a los empleados registrados, listando sus datos.

La opción 6 solicitará un índice al usuario y mostrará los datos de la persona que ocupa esa posición en el registro. Validará que el índice esté dentro de los límites del registro.

La opción 7 permitirá combinar dos registros en uno solo. Fusionará el objeto `registro` con otro objeto `otro_registro` del añadirá 2 valores utilizando el método de `agregar_persona()`.

Reutilice las funciones del módulo Utils.py creado en la práctica anterior para el menú y la lectura de las opciones de menú. Añada una nueva función *leer_cadena(mensaje)* que solicite al usuario que introduzca una cadena y valide que no esté vacía.