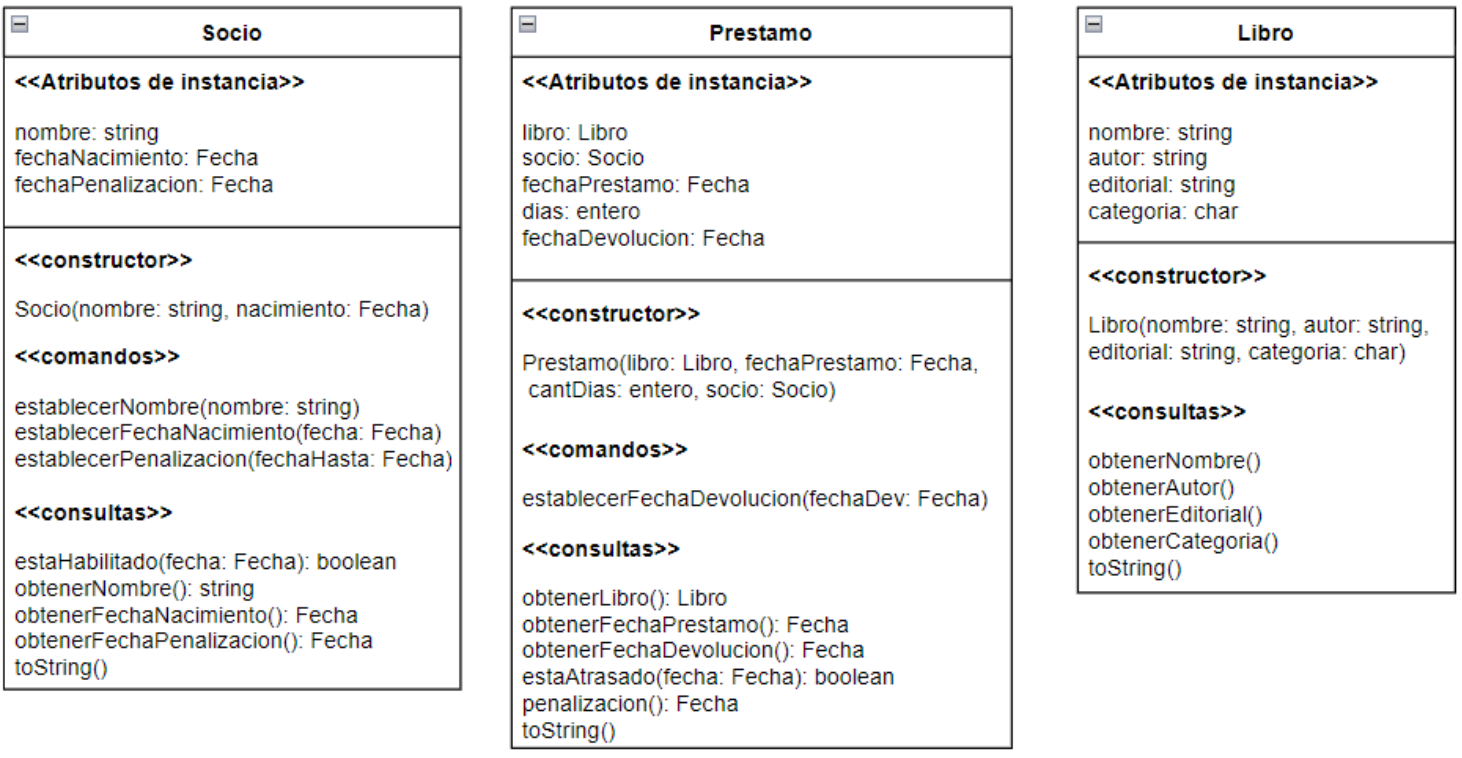
Trabajo práctico N°6

Clases y objetos - relaciones - identidad, igualdad y equivalencia

1) En una Biblioteca se tiene un sistema que mantiene información de los libros, los socios, y los préstamos que se realizan. De los libros se mantiene el nombre, su autor, la editorial y la categoría a la que pertenecen, que puede ser A, B o C. De los socios se mantiene el nombre, la fecha de nacimiento y la fecha de penalización (si es que hubiere) indicando hasta qué fecha no puede retirar libros.
Un socio puede retirar libros, cuando se realiza el préstamo se almacena al socio, el libro, la fecha en que se realiza el préstamo y la cantidad de días que se autoriza el préstamo. Cuando el socio devuelve el libro se almacena en el préstamo la fecha de la devolución y se revisa si el préstamo fue devuelto a tiempo o si se excedió en el plazo. En el caso que el libro se haya devuelto fuera del plazo permitido se calcula una penalización al socio, que puede ser de 3 días si se atrasó menos de una semana (menos de 7 días), 5 días si se atrasó menos de tres semanas (menos de 21 días) y 10 días si se atrasó 3 semanas o más(21 días o más). Además, si el libro tiene categoría A los días de penalización se duplican.
El préstamo de un libro se modela de acuerdo al siguiente diagrama (*se utiliza la clase Fecha del práctico 5*).



En la clase Socio:

- el constructor inicializa al socio sin fecha de penalizacion (None)
- establecerFechaPenalizacion(fechaHasta: Fecha) actualiza el atributo de instancia fechaPenalizacion.

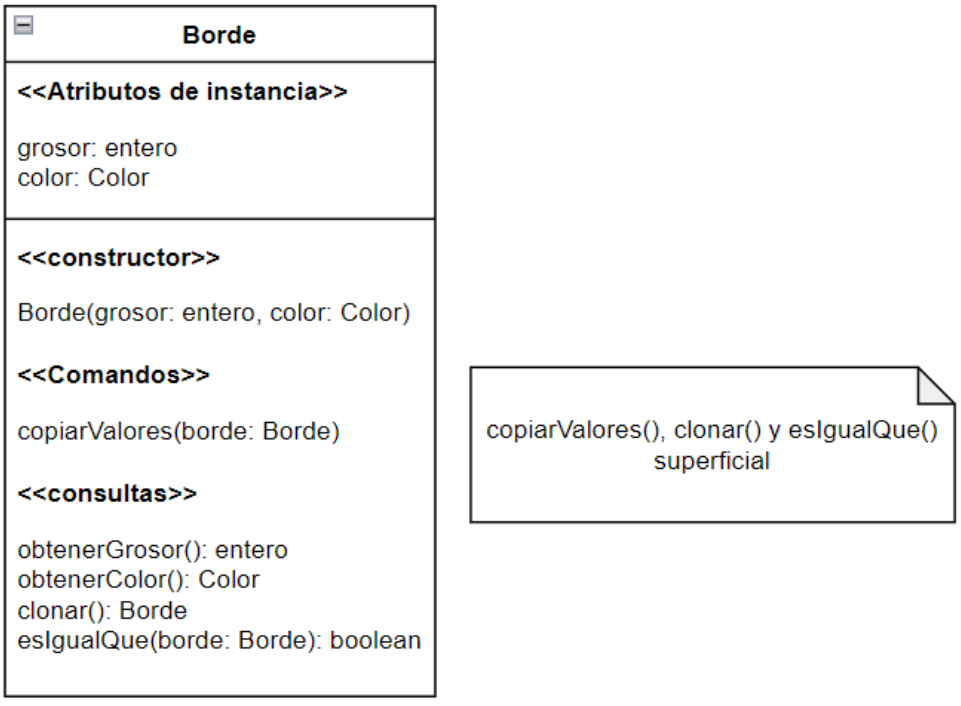
- la Consulta estaHabilitado(fecha: Fecha) retorna True cuando no tiene fechaPenalizacion o cuando ésta es anterior a la fecha recibida en el parámetro.

En la clase Prestamo:

- establecerFechaDevolucion recibe como parámetro la fecha en la que efectivamente se realizó la devolución del libro, y controla si el socio debe recibir una penalización, en caso afirmativo se le asigna al socio la fecha de penalización.
- obtenerFechaDevolución retorna la fecha en la que efectivamente se realizó la devolución del libro.
- estaAtrasado recibe como parámetro la fecha actual y retorna verdadero si el libro no se devolvió y ya debería haberse devuelto de acuerdo a la fecha de préstamo y la cantidad de días.
- penalizacion calcula la cantidad de días de penalización y devuelve la fecha hasta la que corresponde aplicar la penalización, a partir de la fecha de devolución, que tiene que estar ligada. La penalización es de 3 días si se atrasó menos de una semana, 5 días si se atrasó menos de tres semanas y 10 días si se atrasó 3 semanas o más. Si el libro tiene categoría A los días de penalización se duplican.

A. Implemente el diagrama completo.
B. Implemente una clase tester para verificar los servicios de la clase Préstamo, algunos casos con valores fijos y otros casos pidiendo datos al usuario.

2) Dada la clase Color implementada en el práctico 5, implemente la clase Borde de acuerdo al siguiente diagrama. El borde tiene un grosor y es de un determinado color. Cuando se crea un borde recibe un valor para el grosor y un color. Tiene los comandos y consultas triviales, y agrega:
   - un comando *copiarValores* que le permita copiar los valores de otro borde
   - una consulta *clonar* que retorna un clon del borde
   - una consulta *esIgualQue* que devuelve True si el borde recibido es igual a la instancia que recibe el mensaje



3) Dada la implementación de las clases Color y Borde, y el siguiente fragmento de código:

```
C1 = Color(150, 150, 150)
C2 = Color(150, 150, 150)
B1 = Borde(1,C1)
B2 = Borde(1,C2)
print(C1 == C2)
print(B1 == B2)
print(C1.esIgualQue(C2))
print(B1.esIgualQue(B2))
```

   A. Dibuje el diagrama de objetos al terminar el bloque de asignaciones.
   B. Muestre la salida.

4) Dada la implementación de las clases Color y Borde, y el siguiente fragmento de código:

```
C1 = Color(150, 150, 150)
C2 = Color(150, 150, 150)
B1 = Borde(1,C1)
B2 = Borde(1,C2)
B3 = B2.clonar()
B4 = Borde(B2.obtenerGrosor(), B2.obtenerColor())
print( B2 == B3)
print( B2 == B4)
print(B2.esIgualQue(B1))
print(B2.esIgualQue(B3))
print(B2.obtenerGrosor() == B1.obtenerGrosor() and
B2.obtenerColor() == B1.obtenerColor())
print(B2.obtenerGrosor() == B3.obtenerGrosor() and
B2.obtenerColor() == B3.obtenerColor())
print(B2.obtenerGrosor() == B1.obtenerGrosor() and
B2.obtenerColor().esIgualQue(B1.obtenerColor()))
print(B2.obtenerGrosor() == B4.obtenerGrosor() and
B2.obtenerColor().esIgualQue(B4.obtenerColor()))
```

5) Muestre la evolución del diagrama de objetos y la salida del ejercicio anterior considerando que la clase Borde implementa:
   A. clonación en profundidad, igualdad superficial
   B. clonación superficial, igualdad en profundidad
   C. clonación en profundidad, igualdad en profundidad

6) Un club de deportes está organizando un torneo de atletismo, y nos pide que le ayudemos a gestionar la información de los participantes y las disciplinas. El sistema debe permitir registrar a los participantes y las disciplinas en las que compiten.

   Cada Participante tiene un nombre, una edad y una nacionalidad.

   Un Participante puede competir en varias Disciplinas.

   Cada Disciplina tiene un nombre (como carreras de 100 metros, carreras de 400 metros, salto de longitud, salto en alto, lanzamiento de jabalina, etc.) y una descripción (que explica las reglas o características de esa disciplina).

   Varios Participantes pueden competir en la misma Disciplina, y un Participante puede competir en varias disciplinas.

   Se pide:

   A. Diseñar el diagrama UML de las clases.
   B. Implementar las clases en python.
   C. Desarrollar una clase tester que pida al usuario los datos para registrar las disciplinas, y los datos de los participantes. Luego debe permitir al usuario ver todos los participantes inscriptos en una disciplina y las disciplinas en las que participa cada competidor.

7) Un refugio de animales nos pide diseñar un sistema simple para gestionar las mascotas en el refugio. El sistema permitirá registrar mascotas y sus cuidadores, así como asignar y gestionar el cuidado de las mascotas.

   Mascota: Cada mascota tiene un nombre, una especie (por ejemplo, perro, gato), una edad y una descripción. Las mascotas pueden ser asignadas a un cuidador.
   Cuidador: Cada cuidador tiene un nombre, una dirección y un número de teléfono. Los cuidadores pueden ser asignados a una o más mascotas.

   A. Crea un diagrama de clases que incluya dos clases principales: Mascota y Cuidador.

      La clase Mascota debe tener los atributos: nombre, especie, edad, descripción.

      La clase Cuidador debe tener los atributos: nombre, dirección, teléfono y un método para asignar mascotas.

   B. Implementar las clases con python

8) Un organizador de eventos nos pide diseñar un sistema simple para gestionar los eventos. El sistema debe permitir registrar eventos, participantes y organizadores, y asignar participantes a eventos y organizadores a eventos.

Requerimientos

Evento: Cada evento tiene un nombre, una fecha y una descripción. Los eventos pueden tener múltiples participantes y un organizador asignado.

Participante: Cada participante tiene un nombre, una dirección de correo electrónico y un número de teléfono. Los participantes pueden registrarse en uno o más eventos.

Organizador: Cada organizador tiene un nombre, una dirección de correo electrónico y una especialidad (por ejemplo, planificación de eventos, catering, músico, DJ, etc.). Cada organizador puede estar a cargo de uno o más eventos.

A. Crea un diagrama de clases que incluya tres clases principales: Evento, Participante y Organizador.
B. Implementa las clases en python
C. Crea una clase tester para cada clase implementada

9) Un parque de diversiones desea diseñar un sistema para gestionar las aventuras en el parque. El sistema permitirá gestionar atracciones, visitantes, entradas y guías de aventura.

Requerimientos
Atracción: Cada atracción tiene un nombre, un tipo (por ejemplo, montaña rusa, casa embrujada), un nivel de emoción (bajo, medio, alto) y una estatura mínima requerida (por ejemplo: 1,40). Las atracciones pueden funcionar en uno o más turnos (“mañana”, “tarde” o “noche”).

Visitante: Cada visitante tiene un nombre, una edad, una altura y una dirección de correo electrónico. Los visitantes pueden comprar entradas y disfrutar de las atracciones, y llevan un registro de las atracciones a las que pudieron ingresar.

Entrada: Cada entrada tiene un número de entrada, una fecha y un tipo (por ejemplo, entrada general, pase VIP). Cada entrada está asociada a un visitante.

Guía de Aventura: Cada guía tiene un nombre y un turno de trabajo (“mañana”, “tarde” o “noche”). Los guías deben autorizar al visitante cada vez que desee utilizar una atracción en base a su estatura.

- Crea un diagrama de clases que incluya las cuatro clases principales: Atracción, Visitante, Entrada y Guía de Aventura.
- La clase Atracción debe tener atributos como nombre, tipo, estatura mínima, nivel de emoción y turnos de funcionamiento.
- La clase Visitante debe tener atributos como nombre, edad, estatura y correo electrónico.
- La clase Entrada debe tener atributos como número de entrada, fecha y tipo, y debe estar asociada a un Visitante.
- La clase Guía de Aventura debe tener atributos como nombre y turno de trabajo.