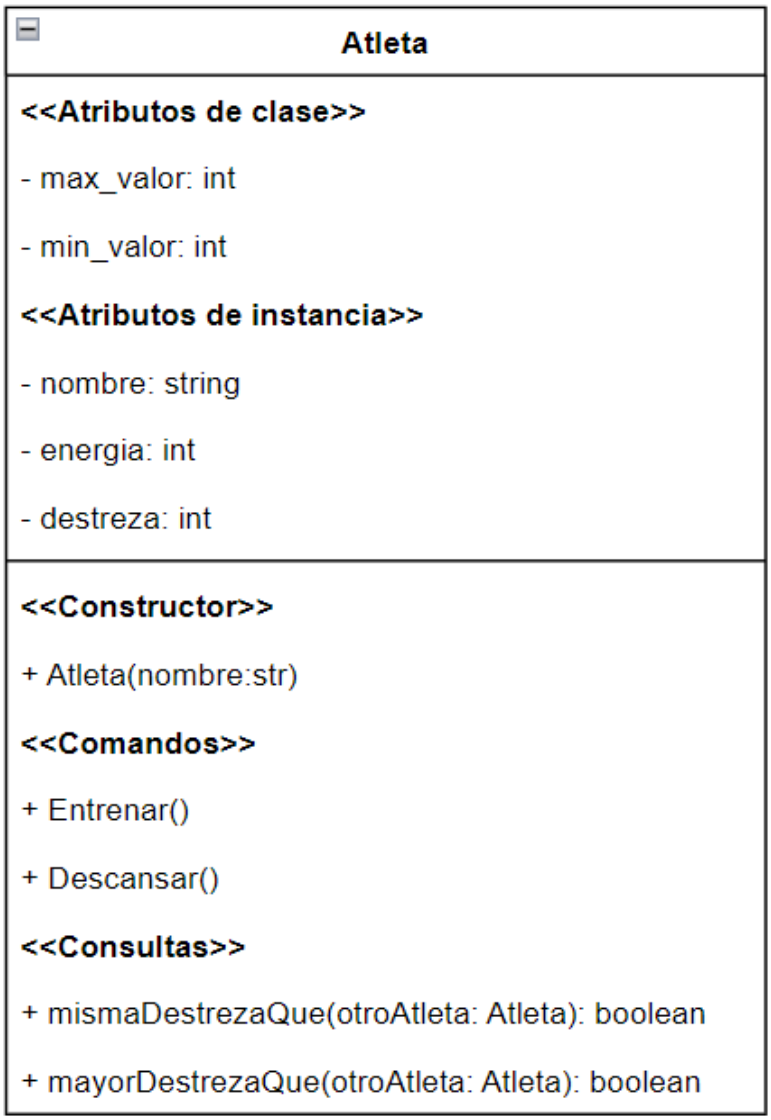
Trabajo práctico N°5

1) En un videojuego se modelan atletas que tienen la capacidad de entrenar para mejorar su destreza o descansar. Cada atleta tiene un nivel de energía que comienza en un valor máximo predeterminado al crearse. Cuando un atleta entrena, consume 5 unidades de energía, y cuando descansa recupera 20. Cada cinco entrenamientos el atleta mejora en 1 punto su destreza, que al crearse comienza con su valor en el mínimo predeterminado. La
Los servicios mismaDestrezaQue y esMejorQue comparan a los atletas por su nivel de destreza.

    a) Implemente el siguiente diagrama de acuerdo a la especificación, encapsulando atributos y comportamiento. Debe implementar también los comandos y consultas triviales que no figuran en el diagrama.

    b) Implemente una clase tester que verifique los servicios de la clase atleta.



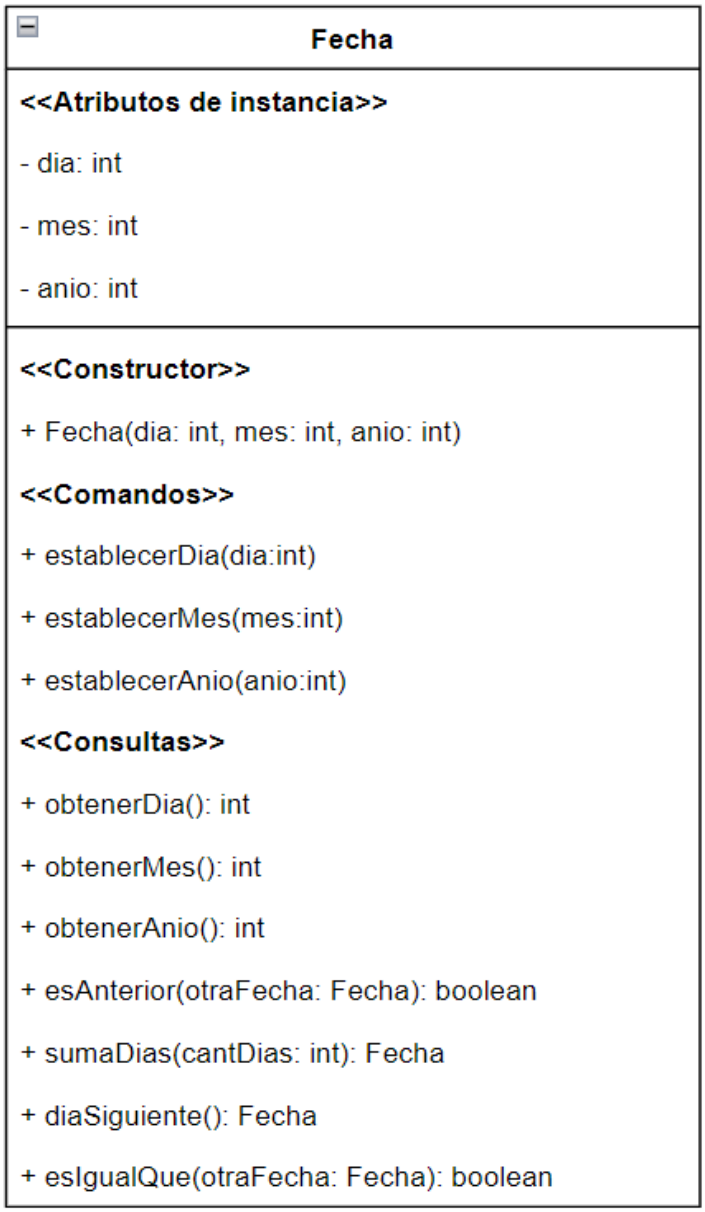
2) A partir del siguiente diagrama que modela la clase Fecha implemente y verifique la clase en python encapsulando atributos y comportamiento. Considerando que:

   a) esAnterior retorna verdadero si la fecha que recibe el mensaje es anterior a la fecha pasada por parámetro, y falso en caso contrario.

   b) sumaDias retorna la fecha que resulta de sumar la cantidad de días recibida por parámetro a la fecha que recibe el mensaje

   c) diaSiguiente retorna una nueva fecha con los valores del día siguiente a la fecha que recibe el mensaje

   d) esIgualQue retorna true si otraFecha es equivalente a la fecha que recibe el mensaje



3) El sistema de colores RGB (Red, Green, Blue) es un modelo de colores aditivos que combina estos tres colores primarios para crear una amplia gama de colores. Cada color en el modelo RGB se representa con tres componentes:
    a) **Rojo (R)**: Intensidad de la luz roja (de 0 a 255).
    b) **Verde (G)**: Intensidad de la luz verde (de 0 a 255).
    c) **Azul (B)**: Intensidad de la luz azul (de 0 a 255).

    Cuando se combinan diferentes intensidades de estos colores, se puede formar cualquier color visible. Por ejemplo:
    d) **Blanco** se forma con R=255, G=255, B=255.
    e) **Negro** se forma con R=0, G=0, B=0.
    f) **Gris** tiene valores iguales para R, G, y B (por ejemplo, R=50, G=50, B=50).

    La combinación y variación de estos valores permite definir colores precisos para una variedad de aplicaciones, como pantallas y gráficos.

Color
<<atributos de instancia>> - rojo: entero - verde: entero - azul: entero
<<Constructores>> + Color() + Color(rojo: entero, verde: entero, azul: entero) **<<Comandos>>** + variar(valor:entero) + variarRojo(valor:entero) + variarAzul(valor:entero) + variarVerde(valor:entero) + establecerRojo(valor:entero) + establecerAzul(valor:entero) + establecerVerde(valor:entero) + copiar(otroColor: Color) **<<Consultas>>** + obtenerRojo(): entero + obtenerVerde(): entero + obtenerAzul(): entero + esRojo(): boolean + esGris(): boolean + esNegro(): boolean + complemento(): Color + esIgualQue(otroColor:Color): boolean + clonar(): Color

Color() inicializa los atributos rojo, verde y azul en 255

- **variar(val:entero)**, modifica cada componente de color sumándole si es posible, un valor dado. Si sumándole el valor dado a una o varias componentes se supera el valor 255, dicha componente queda en 255. Si el argumento es negativo la operación es la misma pero en ese caso el mínimo valor que puede tomar una componente, es 0.
- **variarRojo(val:entero)**, modifica la componente de rojo sumándole un valor dado. Ídem para azul (variarAzul(val:entero)) y verde (variarVerde(val:entero)).
- **esRojo()** retorna el valor verdadero si el objeto que recibe el mensaje representa el color rojo. Ídem para gris (esGris()) y para negro (esNegro())
- **complemento()** retorna un nuevo objeto con el color complemento del color del objeto que recibe el mensaje para alcanzar el color blanco.
- **esIgualQue(otroColor:Color)**: retorna el valor verdadero si ambos objetos son equivalentes.
- **clonar()**: devuelve un nuevo color con el mismo estado interno que el color que recibe el mensaje.

A partir del diagrama presentado y la especificación, implemente y verifique la clase Color en Python encapsulando atributos y comportamiento para modelar la situación planteada. Incluya todos los métodos auxiliares que considere oportunos.

4) En base a la clase Color definida en el punto anterior y considerando la siguiente secuencia de instrucciones:

```
color_1 = Color()
color_2 = Color(70, 70, 70)
color_3 = Color(255, 255, 255)
igualdad1 = color_1.esIgualQue(color_2)
igualdad2 = color_2.esIgualQue(color_3)
color_4 = color_1
color_5 = color_2.clone()
```

   a) elabore un diagrama de objetos que muestre la evolución de las referencias

   b) continúe el diagrama a partir de las siguientes instrucciones:

```
color_6 = Color(140,250,140)
color_4 = color_6
color_2 = color_5.clone()
igualdad3 = color_2.esIgualQue(color_5)
color_3 = color_2
color_1 = color_5.complemento()
```

   c) Explique qué sucede en las siguientes instrucciones:

```
color_1 = color_5.complemento()
color_2 = color_5.clone()
color_3 = color_2
```

5) Una organización dedicada a la preservación del medio ambiente mantiene información referida a cada especie animal de la que se realiza seguimiento. Cada especie está caracterizada por su nombre científico, la cantidad de hembras y la cantidad de machos que se registran en la actualidad y el ritmo de crecimiento anual de la población de acuerdo a lo ocurrido en los últimos 10 años. La cantidad de hembras y machos tiene que ser siempre un valor no negativo, el ritmo puede ser cualquier valor real.
Los tres valores numéricos se inicializan en 0 y luego pueden establecerse o actualizarse con un valor dado. Los comandos establecerMachos y establecerHembras controlan que el parámetro sea positivo, en caso contrario no modifican el valor del atributo. Los comandos actualizarMachos y actualizarHembras suman el valor del parámetro al atributo de instancia. Pueden recibir un parámetro negativo, pero el valor del atributo nunca debe ser menor a 0.
La consulta clonar retorna una especie con el mismo estado interno que la especie que recibe el mensaje. Observe que luego de crear una especie debe establecer los valores de modo que el estado interno de la especie que retorna sea el mismo que el de la especie que recibe el mensaje.
La organización está interesada en calcular para una especie particular: la población total, el nivel de riesgo de extinción (“verde” si el ritmo de crecimiento es positivo, “amarillo” si es nulo y “rojo” si es negativo), la población estimada para dentro de una cantidad dada de años, cuántos años serán necesarios estimativamente para que la población alcance un valor dado. Se desea determinar también si en una especie dada es mayor el número de hembras o de machos y entre dos especies diferentes cuál tiene mayor ritmo de crecimiento.

   a) A partir del diagrama y la especificación implemente la clase Especie en Python encapsulando atributos y comportamiento para modelar la situación planteada. Incluya todos los métodos auxiliares que considere oportunos, como así también los métodos triviales que no estén incluidos en el diagrama.

   b) Escriba una clase tester que verifique los servicios provistos por Especie con valores ingresados por el usuario para nombre y ritmo y generados al azar para machos y hembras dentro de un rango establecido

Especie
- nombre: string - machos: entero - hembras: entero - ritmo: real
<<constructor>> + Especie(nombre: string) **<<comandos>>** + establecerHembras(cantHembras: entero) + establecerMachos(cantMachos: entero) + establecerRitmo(ritmo: real) + actualizarHembras(cantHembras: entero) + actualizarMachos(cantMachos: entero) + actualizarRitmo(ritmo: real) **<<consultas>>** + poblacionActual(): entero + poblacionEstimada(anios:entero): entero + aniosParaPoblacion(poblacion:entero): entero + riesgo(): String + masHembras(): boolean + mayorRitmo(otraEspecie: Especie): Especie + clonar():Especie + toString(): String

mayorRitmo() devuelve la referencia al objeto con mayor ritmo de crecimiento

Clonar(): devuelve un nuevo objeto Especie con el mismo estado interno el objeto que recibe el mensaje