PRACTICO N° 4

EJERCICIO 1. Dado el siguiente diagrama:

Fecha
<<Atributos de instancia>> dia: entero mes: entero anio: entero
<<Constructor>> Fecha(d,m,a:entero) **<<Comandos>>** establecerDia(d:entero) establecerMes(m:entero) establecerAnio(a:entero) **<<Consultas>>** obtenerDia(): entero obtenerMes(): entero obtenerAnio(): entero esAnterior(f:Fecha):booleano sumaDias(d:entero):Fecha diaSiguiente():Fecha

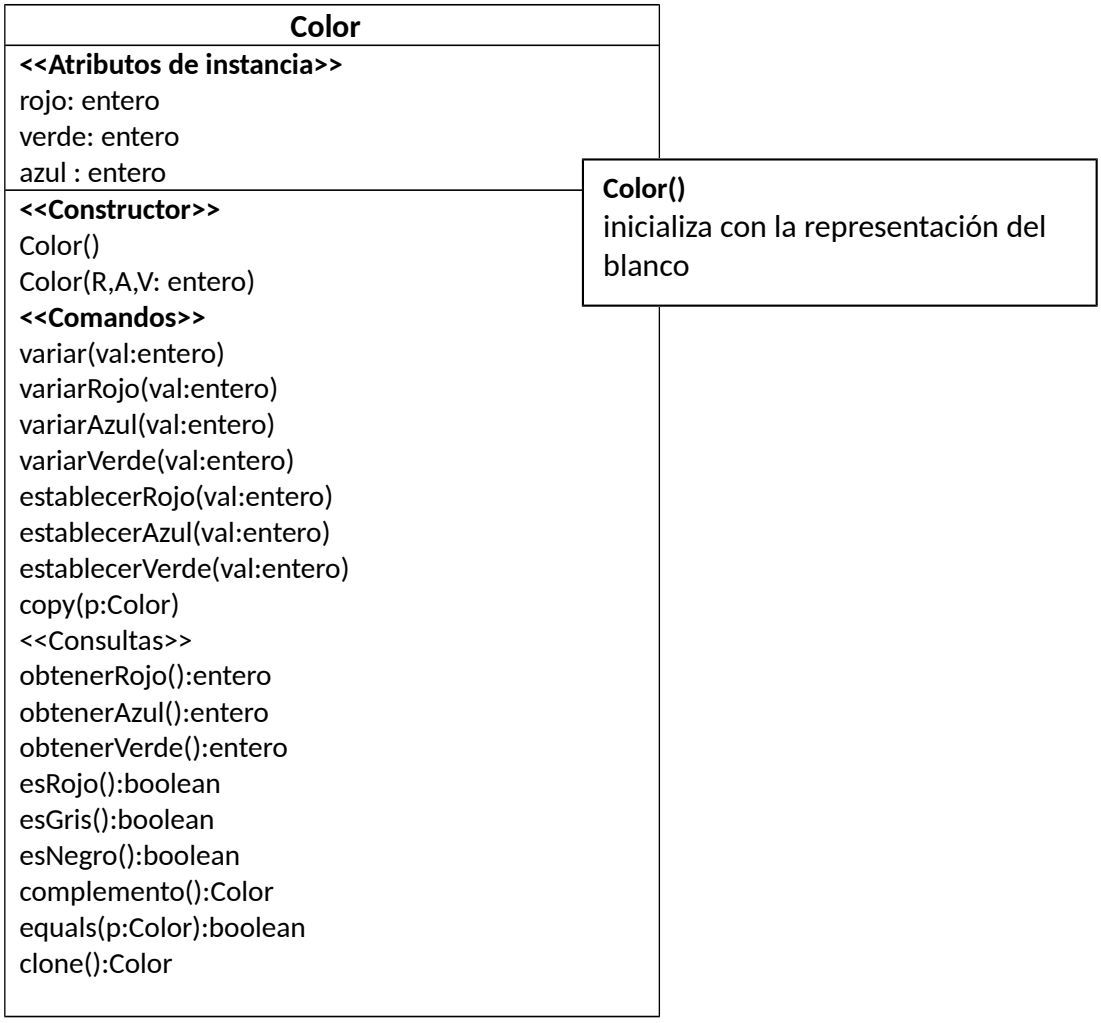
`sumaDias(d)` requiere d<28.

donde

- **esAnterior** retorna verdadero si la fecha que recibe el mensaje es anterior a la fecha pasada por parámetro, y falso en caso contrario.
- **sumaDias** retorna la fecha que resulta de sumar d días a la fecha que recibe el mensaje.

A partir del diagrama presentado y la especificación, **implemente** y **verifique** la clase *Fecha* en Java encapsulando atributos y comportamiento.

EJERCICIO 2. Cuando observamos una imagen en la computadora no percibimos cada color particular, pero si aumentamos la imagen, por ejemplo un 600%, cada color va a ser distinguible. La percepción humana de la luz también es muy limitada y depende de nuestros sensores del color.
Nuestro cerebro determina que color “ve” en base a sensar tres colores: azul, verde y rojo. De modo que cada color puede codificarse mediante una terna de números, el primero representa la cantidad de color rojo, el segundo la cantidad de color verde y el tercero la cantidad de color azul. El rango para cada valor es 0..255. Combinando el máximo de los tres se obtiene el blanco (255, 255, 255). La ausencia de los tres produce el negro (0, 0, 0). El rojo es claramente (255, 0, 0). Cuando se mantiene el mismo valor para las tres componentes se obtiene gris. La terna (50, 50, 50) representa un gris oscuro, (150, 150, 150) es un gris más claro. Esta representación se llama modelo RGB.



- ***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.

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

EJERCICIO 3 Dada la clase Color definida antes y considerando las siguientes secuencias de instrucciones:

```
Color S1,S2,S3,S4;
boolean b1,b2;

S1 = new Color(100,90,120);
S2 = new Color(55,110,100);
S3 = new Color(110,110,100);
S4 = S1;
S1 = S3;
S2 = S1;
S1 = new Color(80,80,80);
```

a) Elabore un diagrama de objetos que muestre la evolución de las referencias.
b) Continúe el diagrama a partir de las siguientes secuencias de instrucciones y muestre los valores de b1 y b2 cada caso:

```
1. S2 = S1.clone();          4. S3.copy(S1);            7. S4 = S1;
2. b1 = S1==S2;              5. b1 = S1==S3;            8. b1 = S1==S4;
3. b2 = S1.equals(S2);       6. b2 = S1.equals(S3);     9. b2 = S1.equals(S2);
```

c) Indique qué sucede al agregar las instrucciones:

```
S1 = null;
S2 = S1.clone();
```