IIP Primer Parcial - ETSInf

10 de Noviembre de 2014. Duración: 1 hora y 30 minutos.

Se desea hacer una aplicación para representar un juego con bloques de distintos colores y dimensiones, apilables en torres. Cada uno de estos bloques tiene asociado un **color** (azul o rojo) y una **dimensión** (número entero entre 1 y 50, ambos incluidos).

Las reglas de este juego indican que:

- Los bloques apilados en una torre deben seguir colores alternos (encima de un bloque azul solo puede haber un bloque rojo y viceversa).
- Encima de un bloque de dimensión x solo puede haber un bloque de dimensión y, donde $y \leq x$ (la torre se estrecha hacia la punta, es decir, se ensancha hacia la base).
- Un bloque puede ser un **comodín**, en cuyo caso puede ir encima de cualquier otro bloque independientemente de su color. Ahora bien, un bloque comodín debe de respetar, como cualquier otro, la regla de la dimensión.

1. 6 puntos Se pide implementar la clase `Bloque`, para lo que se debe:
   a) (0.5 puntos) Definir los atributos de clase públicos y constantes que representan los dos colores posibles de un bloque: `AZUL` y `ROJO`, con valores enteros 0 y 1, respectivamente. Estos atributos deberán ser utilizados siempre que se requiera (tanto en la clase `Bloque` como en la clase `TorreBloques`).
   b) (0.5 puntos) Definir los atributos de instancia privados `color` (`int`), `dimension` (`int`), `comodin` (`boolean`).
   c) (1.5 punto) Implementar dos constructores:
      - Un constructor general con los parámetros apropiados para inicializar todos los atributos de instancia.
      - Un constructor por defecto que crea un `Bloque` azul que no es comodín y cuya dimensión se determina aleatoriamente dentro del rango [1, 50].
   d) (0.5 puntos) Escribir el método consultor y el método modificador del atributo `dimension`.
   e) (1 punto) Escribir el método `equals` (que sobrescribe el de `Object`) para comprobar si un bloque es igual a otro, i.e. si otro es también un bloque y los atributos de uno y otro coinciden uno a uno.
   f) (1 punto) Escribir el método `toString` (que sobrescribe el de `Object`) para que el resultado tenga un formato como el mostrado en los siguientes ejemplos: "`(Color: rojo, dimensión: 22 y SÍ es comodín)`", "`(Color: azul, dimensión: 15 y NO es comodín)`".
   g) (1 punto) Escribir un método `puedeEstarEncimaDe(Bloque b)` que compruebe si un bloque puede, aplicando las reglas ya mencionadas, estar encima de otro `b` que recibe como argumento. Por ejemplo, dadas dos variables `a` y `b` de tipo `Bloque`, `a.puedeEstarEncimaDe(b)` será cierto si y solo si la dimensión del bloque `a` es menor o igual que la del bloque `b` y, o bien `a` es un comodín, o bien los colores de `a` y `b` son distintos.

2. 4 puntos Se pide implementar el programa `TorreBloques`, cuyo objetivo es poder hacer pruebas con pequeñas torres de algunos bloques. Para ello, se pide implementar dicha clase con un método `dimensionValida` que:
   a) (1.25 puntos) limite el valor de la dimensión `d` que se le pasa como parámetro a uno de los valores del intervalo [1, 50]. Esto es: si `d` es menor que 1, devuelve 1; si `d` es mayor que 50, devuelve 50; en cualquier otro caso, devuelve `d` sin modificarlo.

   y con un método `main` que realice las siguientes acciones:
   a) (0.25 puntos) Crear un `Bloque b1` con el constructor por defecto.
   b) (0.5 puntos) Crear un `Bloque b2` de color azul, dimensión 30 y que sea comodín (con el constructor general).
   c) (1 punto) Tras leer su color y dimensión desde teclado, crear un `Bloque b3` que no sea comodín.
      El color se solicitará al usuario como un `String` con valores "rojo" o "azul"; si el usuario introduce cualquier otro valor, el color del `Bloque` será rojo.
      El valor de dimensión se solicitará al usuario como un entero en [1,50] y, por si el usuario introduce cualquier otro valor, se deberá invocar al método `dimensionValida` para garantizarlo.
   d) (0.25 puntos) Mostrar por pantalla los tres objetos creados.
   e) (0.75 puntos) Comprobar si se puede formar una torre situando el bloque `b3` encima del `b2` y este último encima del bloque `b1`. Después, mostrar por pantalla un mensaje con el resultado de dicha comprobación, i.e. si se ha podido formar tal torre o no.

   Se deben importar las clases que se consideren necesarias y utilizar las constantes definidas en la clase `Bloque` donde sea oportuno.