Ejercicios adicionales UT4 (III) – *Interacción de objetos y bucles*

Ejercicio 8. Abre el proyecto AD 08 DemoDados AL y completa la clase DemoDado:

- define un atributo *dado*
- en el constructor crea el dado
- completa el método test1() que simula el lanzamiento de un dado TOTAL veces (TOTAL es una constante) y cuenta y visualice las apariciones del 1 y del 6 en esos lanzamientos
- completa el método test2() que lanza el dado hasta que salga el 4 o el 5 contando las veces que se ha tirado el dado. Visualiza lo que va saliendo en cada tirada.
- completa el método test3() que tira el dado mientras la suma de las tiradas no supere el valor 42. Visualiza lo que vale la suma de tiradas después de cada tirada.

Ejercicio 9. Modifica el método iniciar() de la clase InterfazTexto del ejercicio 7 de manera que ahora el método implemente un bucle. Así, mientras el usuario no pulse la opción SALIR se le estará mostrando continuamente el menú para que elija opción. Valida además la opción comprobando que es un valor entre 1 y 5.

Ejercicio 10. Añade a la clase Fraccion del ejercicio 6 un método privado simplificar() sin parámetros y sin valor de retorno que simplifica la fracción (el numerador y denominador del objeto actual). Para simplificar se utilizará otro método privado mcd() cuya signatura es:

```
private int mcd(int  num1, int num2)
```

y que devuelve el máximo común divisor de dos números. Para calcular el máximo común divisor de dos valores se utilizará el *algoritmo de Euclides*:

> *se dividen ambos n^{os} (el dividendo es el mayor de los dos) calculando el resto. Si el resto es 0 entonces el mcd es el último divisor. Si el resto no es 0 se repite el proceso, pero ahora el dividendo es el divisor y el divisor es el resto.*

Modifica el constructor con parámetros de la clase Fraccion para que siempre quede simplificada en el momento de su construcción.

Ejercicio 11. En este ejercicio crearás una aplicación que permita hacer operaciones con números exponenciales de idéntica base (asumimos esto último).

El número exponencial $\mathbf{a^b}$ significa que la base $\boldsymbol{a}$ se está multiplicando por sí misma tantas veces como indique el exponente $\boldsymbol{b}$. (a y b son nºs enteros). (*Ej.* $4^5=1024$)

El producto de dos n^{os} exponenciales de igual base se define**:** $\mathbf{a^n * a^m = a^{(n+m)}}$
(*Ej.* $4^5*4^2=4^7=16384$)

La potencia de un nº exponencial es: $\mathbf{(a^n)^m = a^{(n*m)}}$ (*Ej.* $(2^3)^2=2^6=64$)

El cociente de números exponenciales de igual base se define: $\mathbf{a^n / a^m = a^{(n-m)}}$
(*Ej.* $\frac{2^5}{2^3}=2^2=4$)

El valor de un nº exponencial es:

$\mathbf{a^{(-n)} = 1 / a^n}$
$\mathbf{a^o = 1}$
$\mathbf{a^n = a*a*a*....}$ **(n veces)**

(*Ej.* $4^{-5}=\frac{1}{4^5}=\frac{1}{1024}=0{,}00097$) (*Ej.* $4^0=1$) (*Ej.* $4^5=4*4*4*4*4=1024$)

La clase Exponencial modela un nº exponencial. Tiene dos atributos enteros, *base* y *exponente*.

La clase incluye:

- dos constructores sobrecargados, uno tiene dos parámetros, *base* y *exponente*. El otro recibe un objeto Exponencial a partir del cual se crea el objeto Exponencial actual.
- accesores y mutadores para *base* y *exponente*
- el método public double valorExponencial() - calcula el valor de un nº exponencial. Para ello este método se ayuda del método privado,
  private int potencia(int a , int b) que calcula a^b (el parámetro *b* recibido por este método siempre es un valor >= 0)

- el método public Exponencial multiplicar(Exponencial otro) que multiplica dos nºs exponenciales
- el método public Exponencial dividir(Exponencial otro)
- el método public Exponencial elevar(int n) que calcula la potencia de un nº exponencial
- el método toString() devuelve una representación textual del nº exponencial que incluya la base, el exponente y el valor exponencial del nº

Añade una clase AppExponencial que incluye el método main(). Incluye en esta clase código de prueba para testear la clase anterior. Hazlo de la siguiente manera:

- define y crea dentro del main() un objeto local *teclado* para leer valores desde el teclado
- pide al usuario valores de *base* y *exponente* y crea un objeto Exponencial
- pide ahora una segunda base validando con un bucle *while* que sea igual a la anterior. Pide un nuevo exponente y crea otro objeto Exponencial
- prueba los métodos de la clase Exponencial para obtener los resultados de la figura
- para testear el método elevar() en lugar de pedir por teclado la potencia genérala como un valor aleatorio entre 2 y 8 (ambos inclusive) utilizando la clase Math.

```
Dame base para el primer número exponencial:
2
Dame exponente para el primer número exponencial:
3
Dame base para el segundo número exponencial:
4
Error, Dame base para el segundo número exponencial:
5
Error, Dame base para el segundo número exponencial:
2
Dame exponente para el segundo número exponencial:
4
```

```
Exponencial 1
        Base:2  Exponente:3
        Valor: 8.0
Exponencial 2
        Base:2  Exponente:4
        Valor: 16.0
Producto
        Base:2  Exponente:7
        Valor: 128.0
Cociente
        Base:2  Exponente:-1
        Valor: 0.5
Potencia 7 de número exponencial 1
        Base:2  Exponente:21
        Valor: 2097152.0
```

Ejercicio 12. Descarga el proyecto *AD12-13 Numero AL* y completa la clase Numero. La clase incluye un atributo entero, *numero*. Tiene tres constructores sobrecargados, uno sin parámetros que inicializa el nº a 0, otro con un parámetro entero y el último recibe un objeto Numero.

La clase ofrece los siguientes servicios (métodos):

- public int factorial() - devuelve el factorial del nº guardado en el objeto.
  El factorial de un nº n es: **n! = n * (n-1) * (n-2)* .....* 3*2*1** El factorial de 0 es 1
  Ej. 5! = 5*4*3*2*1

- public boolean esPrimo() - detecta si el nº es primo o no (un nº es primo si tiene como únicos divisores a él mismo y a la unidad) (Haz una versión del método de forma que en cuanto encuentres un divisor el método ya devuelva *false*)

- public void escribirFigura() - escribe un cuadrado relleno en pantalla de * con tantas filas y columnas como indique el nº guardado en el objeto

- public int contarCifras() - calcula y devuelve la cantidad de cifras que tiene el nº

- public boolean esCapicua() - detecta si el nº es o no capicúa. Para ver si un nº es capicúa calcularemos su inverso y lo compararemos con el original. 12798 no es capicúa porque su inverso 89721 no coincide. 12321 es capicúa porque su inverso 12321 coincide.
  Para hacer este método usa el método privado **int inverso(int numero)** que dado un nº lo devuelve invertido. Genera el inverso sin calcular previamente el nº de cifras del nº.

Testea la clase desde BlueJ.

Ejercicio 13. Añade ahora a la clase Numero los siguiente métodos:

- ✗ public int aBase8() - devuelve el nº que guarda el objeto convertido a base 8. Si numero = 17 devuelve 21

- ✗ public boolean estaCifra(int cifra) – devuelve *true* si la cifra se encuentra en el atributo *número* . Utiliza el método privado estaCifra(int numero, int cifra)

- ✗ public boolean hayCifrasRepetidas() - devuelve *true* si el atributo que guarda el objeto contiene cifras que se repiten. Puedes utilizar como método de ayuda el método privado estaCifra()

- ✗ public int aBase2() - devuelve el nº que guarda el objeto convertido a base 2 (si *numero* = 17 devuelve como entero 10001)

Completa la clase InterfazTexto que permitirá al usuario interactuar con los objetos de la clase Numero.

Diseña esta clase de forma análoga a la realizada en el ejercicio 7 con las modificaciones introducidas en el ejercicio 9.

Completa el proyecto con la clase AplicacionNumero que incluye el main() y crea un objeto *miInterfaz* y llama al método iniciar().

Prueba toda la aplicación.