Ejercicios adicionales UT5 – Colecciones de tamaño fijo:arrays. La clase String

Ejercicio 1 .- Abre el proyecto *AD01Histograma AL* y completa la clase Histograma. Define en ella un atributo *valores* que es un array de enteros de tamaño máximo MAX (MAX es una constante privada estática). En el constructor crea el array de tamaño MAX y asigna a cada elemento un valor aleatorio comprendido entre 5 y 50. Hazlo utilizando un método privado inicializar() sin parámetros que inicializa el array. Este método a su vez utiliza el método privado generarAleatorio() que devuelve un nº aleatorio dentro de ese rango. Añade un método escribirHistograma() que muestra en pantalla el histograma:

```
*******
****************
**********
****************************
**********************
******************
********************************************
```

Define para el carácter * una constante de clase ASTERISCO.

Incluye en la clase un método int[] getValores() que devuelve una copia del array. Haz dos versiones de este método, la primera utilizando System.arraycopy() y la segunda con un método de la clase Arrays.

Añade una clase AppHistograma que incluya el método main() e incluye el código necesario para probar la clase Histograma.

Ejercicio 2.- Completa la clase FrecuenciaDado. La funcionalidad de esta clase va a permitir tirar un dado un nº determinado de veces y contar la frecuencia de aparición de cada una de sus caras. La clase incluye un objeto *dado* de la clase Dado y un atributo adicional *frecuencia* en el que se cuenta la frecuencia de sus caras. Los métodos, además del constructor que crea e inicializa adecuadamente los atributos, son:

- public void tirarDado(int veces) – tira el dado las veces que indica el parámetro
- public void escribirFrecuencia() - escribe la frecuencia de aparición de cada una de las caras del dado

Cara	*Frecuencia*
1	*7*
2	*7*
3	*9*
4	*10*
5	*7*
6	*10*

Ejercicio 3.- Abre el proyecto *AD03 Calculadora numeros AL* y completa la clase Calculadora permite guardar una lista de números. La constante de clase MAX indica la máxima cantidad de números que se pueden guardar aunque puede haber menos. Los números se guardan en un array de enteros. El atributo *total* lleva la pista de los números realmente almacenados y sirve además para saber en qué posición se va añadir un nuevo número que llegue a la calculadora.

Hay dos constructores:

- el primer constructor sin parámetros crea el array a su tamaño máximo e inicializa *total* a 0 (al principio no hay números en la calculadora)
- el segundo constructor tiene un parámetro que es un array. El atributo *numeros* de la calculadora se crea a partir de este parámetro. Habrá que comprobar que el tamaño del parámetro recibido no es mayor que MAX. Si es mayor que MAX se crea el atributo a un tamaño igual a MAX, si es menor, al tamaño indicado por el parámetro (usa Math.min()). El atributo *numeros* se crea como una copia del parámetro recibido.

introducirNumero(int) añade un nuevo nº a la calculadora (se añade al final) modificando adecuadamente el valor del atributo *total*. Solo se puede añadir si el array no está completo.

estaLLena() devuelve *true* si la calculadora está completa

El método contarMayoresQue() devuelve la cantidad de valores en la lista mayores que el parámetro recibido.

escribirNumeros() muestra en pantalla la lista de números, en cada línea se visualiza la posición en el array y su valor correspondiente.

El accesor getTotal() devuelve la cantidad de números actualmente almacenados.

vaciar() vacía la calculadora dejándola sin números

intercambiar() intercambia los valores de las posiciones indicadas como parámetros. Dentro de este método se comprobará si las posiciones a intercambiar son correctas con ayuda del método privado posicionCorrecta().

int[] todoDigitosDecreciente() devuelve un array con los números de la calculadora que tengan todos sus dígitos en orden decreciente. Usa el método privado boolean enOrdenDecreciente(int n).

Si *numeros = {45, 6, 965, 123, 93, 489, 321}* devuelve *{6, 965, 93, 321 }*

El método borrarUltimoElemento*()* borra el último elemento de la lista (si esta no está vacía) y actualiza *total*.

El método privado hayNumeros() devuelve *true* si la lista no está vacía, *false* en otro caso.

borrarPares() borra todos los números de valor par que hay en el array y actualiza *total*. Para saber si un nº es par se utilizará el método privado esPar() y para borrar un valor se llamará al método borrarUnPar() donde el parámetro pasado a este método señala la posición del valor a borrar.

insertarEnPosicion(int, int) añade un nuevo nº a la calculadora en la posición indicada. Si la posición es un valor incorrecto se lanza una excepción del tipo IllegalArgumentException. Una posición es incorrecta si no está entre 0 y el total de números guardados en la calculadora. Si la posición es correcta se inserta siempre y cuando el array no esté completo.

```
if  (posicion incorrecta) {
     throw new IllegalArgumentException("Posición incorrecta");
}
```

Añade al proyecto una clase DemoCalculadora que pruebe los métodos anteriores:

- incluye en la clase dos atributos *miCalculadora* y *tuCalculadora* e instáncialos en el constructor
- define el método test1(). Declara dentro de este método un array local e inicialízalo en el momento de la declaración con una serie de valores (no más de MAX). A través de un bucle añade cada uno de los elementos de este array local a *miCalculadora*. Muestra los números de la calculadora, muestra los nºs con todos los dígitos en orden decreciente, borra el último elemento, vuelve a mostrar los números, borra todos los valores pares y muestra el estado final de la calculadora
- incluye otro método test2(). Añade al atributo *tuCalculadora* una serie de 8 valores (genera estos valores con un bucle *for*, duplica el índice del *for* y añade este valor a la calculadora). Muestra la calculadora, escribe cuántos valores mayores que 6 hay actualmente guardados, borra los tres últimos elementos añadidos a la calculadora, muestra el estado final de la misma indicando cuántos valores hay. Vacía la calculadora y muestra la cantidad de valores que quedan
- incluye otro método test3() e inserta en *miCalculadora* un valor en la posición 0, en la posición 1 , en la posición 2 . Muestra el estado final de la calculadora. Prueba también insertando un valor en una posición incorrecta por ejemplo, la posición 33 o -5.

Anota qué dos métodos de la clase Calculadora podríamos haber definido como *static* y por qué.