Ejercicios Java - Unidad 8

Colecciones

Instrucciones para resolver los ejercicios

- Resuelve los ejercicios en el proyecto creado en repositorio “Ejercicios 3ª Evaluación”.
- Las clases definidas para cada ejercicio se han de definir en el paquete `unidad8.colecciones`.
- Cada vez que se resuelva un ejercicio se realizará un *Commit and Push* con el mensaje “Ejercicio *n* de la unidad 8 (colecciones) resuelto”, donde n será el número de ejercicio.
- No es obligatorio resolver y confirmar los ejercicios en el orden de numeración.

Ejercicio 1

Programa que lee de la entrada estándar una secuencia de nombres y los guarda en una colección de forma que ésta no contenga nombres repetidos. La primera línea de entrada contiene la cantidad de nombres. El resto de las líneas contienen los nombres a razón de uno por línea. El programa finalizará mostrando los nombres guardados en la colección en el mismo orden en que se insertaron.

Ejemplo:

Entrada	Salida
6 Iván Iván Iván Manuela Iván Fernando	Ivan Manuela Fernando

Ejercicio 2

Crea un programa que realice las tareas siguientes:

- Almacenar en una lista 100 números aleatorios entre 1 y 100 y mostrarlos todos utilizando uno de los métodos de iteración.
- Almacenar los números de la lista original en otra colección sin que se repita ninguno y mostrarlos todos usando un método de iteración diferente al anterior.
- Almacenar los números de la lista original en otra colección en la que se almacenen ordenados y sin que se repita ninguno y mostrarlos todos usando un método de iteración diferente al anterior.

Ejercicio 3

Crea un programa que lea de la entrada estándar una línea de texto y a continuación almacene en una colección las palabras que no se repiten y en otra colección las que sí se repiten. El programa finalizará mostrando el contenido de ambas colecciones (hacerlo sin escribir código para iterar).

Ejercicio 4

Definir una clase `Palabras` para almacenar palabras según las especificaciones siguientes:

- No se almacenarán palabras repetidas.
- Las palabras se almacenarán en una estructura de datos que facilite la obtención de la lista de palabras de una determinada longitud en orden alfabético.

- Constructores: un constructor sin parámetros que cree la estructura vacía y un constructor que declare un parámetro de tipo `String`, cree la estructura y almacene en ella las palabras que formen parte del parámetro.
- Métodos para:
  - Añadir una palabra.
  - Añadir las palabras que formen parte de una cadena de caracteres.
  - Comprobar si una palabra está contenida en la estructura de datos.

Escribir un programa que ponga a prueba la clase Palabras. El programa mostrará en la consola un indicador para que el usuario introduzca los comandos siguientes (cada comando finaliza con la pulsación de la tecla intro):

- `añadir: …` Se almacenarán las palabras escritas a continuación de los dos puntos.
- `lista n` Se mostrará la lista de palabras de longitud *n*.
- `borrar` Borra todas las palabras almacenadas.
- `borrar: …` Sustituye las palabras almacenadas anteriormente por las palabras escritas a continuación de los dos puntos.
- `fin` Borra todas las palabras almacenadas.

Ejercicio 5

Programa para practicar las operaciones básicas con una cola. Tendrá que leer de la entrada estándar los datos siguientes:

- Una línea en la que se escribirán 3 números enteros que llamaremos *N*, *K* y *X*.
- Una línea en la que se escribirán *N* números, entre los que estará el número *X*.

Los *N* números leídos en la segunda línea se almacenarán en una cola y a continuación se retirarán *K* números. Finalmente se comprobará si el número *X* aún se encuentra almacenado en la cola. Si es así se mostrará *true* en la salida estándar. En caso contrario se mostrará el número más pequeño de los que permanecen en la cola. Si la cola estuviese vacía se mostrará el valor cero.

Ejemplos:

Entrada	Salida
`5 2 32` `1 13 45 32 4`	`true`

Entrada	Salida
`4 1 90` `90 75 13 420`	`13`

Ejercicio 6

Programa para manejar una lista de contactos telefónicos mediante una serie de comandos de texto, teniendo en cuenta que cada contacto puede tener varios teléfonos. El programa se ejecutará en la consola presentando una línea de comando que comenzará con el carácter `>` seguido de un espacio en blanco para indicarle al usuario que puede escribir uno de los comandos siguientes:

Añadir un contacto:

```
> nombre:teléfono
```

- El dato *teléfono* estará formado únicamente por dígitos decimales.
- No se pondrá límite al número de contactos que se pueden agregar ni a la cantidad de teléfonos que puede tener un contacto, salvo el que impone el tamaño de la memoria.
- Si se especifica un nombre nuevo, se crea un nuevo contacto con el teléfono especificado.
- Si el contacto ya existe, se le añade el teléfono especificado.
- Si el teléfono ya existe para ese contacto, se mostrará el mensaje correspondiente.

Buscar los teléfonos de un contacto:

```
> buscar:nombre
```

En la línea siguiente se mostrarán los teléfonos del contacto separados por comas, o el mensaje correspondiente si éste no existe.

Eliminar un contacto a partir de un nombre:

```
> eliminar:nombre
```

Si el contacto existe, se pedirá confirmación para eliminarlo. Si el contacto no existe, se mostrará el mensaje correspondiente.

Mostrar todos los contactos:

```
> contactos
```

Se mostrarán todos los contactos, cada uno en una línea, en orden alfabético.

Finalizar el programa:

```
> salir
```

El programa mostrará los errores correspondientes si alguno de los comandos no se ajusta al formato especificado. En el caso de comandos del tipo *comando:datos*, se permiten espacios en blanco a derecha e izquierda de los dos puntos y al principio y final de la línea de comando.

Después de la ejecución de cada comando se volverá a mostrar la línea de comando.

Ejemplo:

```
> María:600000001
> Alberto:600000002
> Elena:600000003
> buscar:María
Teléfono: 600000001
> buscar:Fernando
El contacto no existe
> Fernando:600000004
> Elena:600000005
> buscar:Elena
600000003, 600000005
> salir
```

Ejercicio 7

Programa que crea dos conjuntos de números enteros con *n* y *m* elementos respectivamente y muestra en la consola los números que están en ambos conjuntos. El programa crea cada conjunto leyendo los datos en la entrada estándar de la forma siguiente:

- En la primera línea se especifican las cantidades *n* y *m* que indican el número de elementos de cada conjunto.
- En la segunda línea se especifican, separados por espacios en blanco, los elementos de ambos conjuntos, primero los *n* elementos del primer conjunto y después los *m* elementos del segundo conjunto.

Ejemplo:

Entrada	Salida
4 3 1 3 5 7 3 4 5	3 5

Ejercicio 8

Programa que cuenta los puntos obtenidos por varios jugadores en un juego de naipes en el que el mazo de cartas está formado por varias barajas. El programa comienza leyendo de la entrada estándar un número indeterminado de líneas, cada una conteniendo un nombre de jugador y una serie de naipes robados del mazo (un número indeterminado de ellos) que se han de añadir a la mano del jugador. El formato de estas líneas es el siguiente:

```
nombre : RP RP RP … RP
```

Cada *RP* es un naipe en el que el rango *R* es uno de los valores del conjunto {*2*, *3*, *4*, *5*, *6*, *7*, *8*, *9*, *10*, *J*, *Q*, *K*, *A*} y el palo *P* es uno de los valores del conjunto {*S*, *H*, *D*, *C*}. Las letras que representan cada palo son las iniciales de **S**pades (picas), **H**earts (corazones), **D**iamonds (diamantes) y **C**lovers (tréboles).

El nombre de cada jugador puede contener cualquier carácter excepto '*:*'.

El fin de la entrada se indica mediante una línea que contiene la palabra "*fin*".

Se supone que la entrada de datos se va a ajustar al formato descrito y, por tanto, no es necesario realizar una comprobación de errores de entrada.

Una vez guardadas las manos de cada jugador, se calculará su valor según las reglas siguientes:

- Se descartan los naipes repetidos en la mano de cada jugador.
- Cada naipe tiene un valor que se calcula como el valor del rango multiplicado por el valor del palo según las tablas siguientes:

RANGOS	2	3	4	5	6	7	8	9	10	J	Q	K	A
VALOR	2	3	4	5	6	7	8	9	10	11	12	13	14

PALOS	S	H	D	C
VALOR	4	3	2	1

El programa finalizará mostrando el nombre de cada jugador y el valor de su mano con el formato:

nombre **:** ***valor***

Ejemplo:

Entrada	Salida

`María: 2C 4H 9H AS QS`
`Fernando: 3H 10S JC KD 5S 10S`
`Ana: QH QC QS QD`
`Fernando: 6H 7S KC KD 5S 10C`
`Ana: QH QC JS JD JC`
`María: JD JD JD JD JD JD`
`fin`
	María: 167
	Fernando: 175
	Ana: 197

Ejercicio 9

Programa que simula el funcionamiento de una línea de procesado robotizada según las especificaciones siguientes:

- Para cada robot se especifica la cantidad de segundos que tarda en procesar un producto.
- Los productos se procesan en el orden en que se depositan en la línea.
- Cada vez que un robot comience a procesar un producto, mostrará en la consola su nombre, el producto que va a procesar y la hora de comienzo.
- La línea avanza a razón de un producto por segundo.
- Si no hay ningún robot disponible para procesar un producto, éste pasa al final de la línea.
- El programa finaliza cuando se han procesado todos los productos.

Los datos para realizar la simulación se leen de la entrada estándar con el formato siguiente:

- En la primera línea se especifica el nombre de los robots y su tiempo de proceso con el formato: ***`nombreRobot-tiempoProceso;nombreRobot-tiempoProceso;nombreRobot-tiempoProceso;`…***
- En la segunda línea se especifica la hora de comienzo con el formato ***`hh:mm:ss`***.
- En las líneas siguientes se especifica el nombre de un producto.
- La última línea contiene la cadena `"fin"` indicando el final de la entrada de datos.

La simulación no comenzará hasta que se hayan leído todos los datos de la entrada estándar.

Ejemplo:

Entrada	**Salida**

`ROB-15;SS2-10;NX8000-3`
`8:00:00`
`disco duro`
`procesador`
`memoria RAM`
`placa base`
`fin`
	`ROB – disco duro [08:00:01]`
	`SS2 - procesador [08:00:02]`
	`NX8000 – memoria RAM [08:00:03]`
	`NX8000 – placa base [08:00:06]`