Colecciones

123.- Pide por teclado un número. Dicho número indica cuantos elementos se van a pedir a continuación. Almacena esas números en un ArrayList. A continuación:

a) Muestra el contenido de la lista.
b) Intercambia la primera posición con la última.
c) Pide otra lista de números distinta como se indica al principio y añade todos sus elementos a la lista inicial. *Nota: al final sólo debe existir un arrayList con todos los valores juntos.*
d) Calcula la suma de todos los elementos de la lista.
e) Calcula la media aritmética de los elementos de la lista.
f) Muestra todos los valores de la lista que sean menores a la media antes calculada.

124.- Reescribe la clase Pila del **ejercicio 79** ("Lista de Cadenas" en la parte de POO Básica) usando como estructura principal un ArrayList. ¿Que ventajas se obtienen frente a la creación de esa clase usando arrays normales?

125.- Crea un programa que, haciendo uso de un ArrayList de números enteros, muestre por pantalla un menú con las siguientes opciones:

1. *Agregar:* se le pide al usuario que indique un número entero por teclado y mete ese número dentro de la estructura.
2. *Buscar:* se le pide al usuario un número entero e indica si ese número se encuentra en la estructura o no.
3. *Eliminar:* se le pide al usuario un número entero y, si el elemento está en la estructura, lo elimina (y lo indica por pantalla). Si el elemento no está en la estructura, saca un mensaje indicándolo.
4. *Modificar:* se le pide al usuario un número entero y, si el elemento está en la estructura, se solicita otro número para cambiarlo. Si el elemento no está en la estructura, saca un mensaje indicándolo.
5. *Insertar en posición:* se le pide al usuario un número entero y una posición (entero mayor a 0). Mete el número en la posición indicada de la estructura.
6. *Mostrar:* muestra el contenido de la estructura.
7. *Salir:* Sale del programa y muestra un mensaje de despedida.

126.- Crea la clase **persona**. De un apersona necesitamos el DNI, nombre, apellido (uno solo), sexo, edad y peso. Todas las propiedades serán privadas.

La clase tendrá un constructor normal (con todos los argumentos) y uno solo con el DNI (nombre y apellido quedaran como cadena vacía, edad y peso se inicializará a 0.

Además de los getters y setters correspondientes, la clase tendrá los métodos: `toString` y `equals`.

Haciendo uso de la clase persona, implementa la clase **Cola de Supermercado**. En una cola de supermercado, la primera persona que entra en la cola, es la primera que puede abandonarla.

Usa las variables de clase que creas necesarias (todas privadas) e implementa los métodos:

a) `EstaVacia`: devuelve Verdadero o Falso si la cola está vacía o no.
b) `Entrar`: mete una persona en la cola.
c) `Salir`: saca de la cola a la persona que le corresponde. Muestra la información de esa persona.
d) `Primero`: este método devuelve una persona. La persona devuelta será la que está primera en la cola. Nota: No muestra nada por pantalla.
e) `Cantidad`: devuelve el número de personas que hay en la cola.

127.- Haciendo uso de la clase persona del ejercicio anterior, crea la clase **Agenda**, la cual tendrá una lista de personas (arrayList) y los métodos:

- `AniadirPersona`: recibe como argumento una persona y la añade a la lista.
- `BorrarPersona`: recibe como argumento un DNI y ,en el caso de que exista una persona con DNI, la borra. *Nota: borra la primera que encuentre.*
- `BuscarPersona`: recibe como argumento un DNI y muestra los datos de la persona con ese DNI. O un mensaje si ese DNI no está en la lista.
- `MostrarAgenda` (`toString`): muestra todos los datos de cada persona de manera ordenada.
- `OrdenarAgenda`: ordena las personas de la lista por apellidos. *Nota: para este punto debes buscar información sobre el método Collections.sort*

Crea un fichero **main** para probar la clase anterior. Para ello, muestra un menú para poder realizar las acciones antes indicadas y prueba su funcionamiento.

128.- La empresa MEDAC te ha elegido para crear un programa que se encargue de registrar los sueldos de los hombres y las mujeres que trabajan allí, con el fin de detectar si existe brecha salarial entre ambos.
Para ello, el programa pedirá primero el número total de trabajadores de la empresa por teclado (N) y, posteriormente, solicitará información de cada **persona** de la siguiente forma:

- pedirá su DNI
- pedirá su género (0 para varón y 1 para mujer)
- pedirá su sueldo anual en euros.

Esta información debe guardarse en un diccionario. Finalmente, el programa mostrará por pantalla el sueldo medio de cada género.

129.- Crea un programa que vaya solicitando nombres de diferentes alumnos junto con su nota media del curso hasta que el nombre sea igual a "fin" (minúsculas). *Nota: supongamos que no existen nombres repetidos.*
Almacena todos esos datos en una estructura tipo diccionario y muestra cada alumno con su nota en una línea.

A continuación sube un punto la nota de aquellos alumnos cuyo nombre empiecen por 'J' y borra aquellos cuyo nombre empiecen 'A' o por 'D'

Para finalizar, si la estructura no está vacía, muestra solo las notas y la media de todas ellas.

130.- Implementa un programa que cuente el numero de apariciones de cada palabra de un texto. Para ello se debe crear una estructura que almacene cada palabra y el número de veces que aparece en el texto.
Recuerda que un texto puede tener entre 0 y N palabras (siendo N un número desconocido) y que una palabra puede estar escrita en mayúscula, minúscula o combinación de ambas (y sigue siendo la misma palabra)

El texto se va a obtener de un fichero que el programa debe leer. No es importante la ruta de ese fichero (ponlo donde quieras).

Tras eso deben mostrarse todas las palabras del texto una debajo de otra junto con el número de veces que aparece cada una.

Nota: Vamos a suponer que NO hay signos de puntuación en el texto. Sólo letras, números, saltos de linea y espacios en blanco.

131.- Crea la clase ARAS la cual facilitará el trabajo con diccionarios de tipo String-String. Para ello es necesario implementar los siguientes métodos:

- `borrar(String clave)`: elimina la celda indicada por la clave.
- `meter(String clave,String valor)`: introduce una nueva celda en el diccionario con la clave y el valor indicados. Si ya existe una celda con esa clave, lo indica por pantalla y no toca el diccionario.
- `actualizar(String clave,String valor)`: busca la celda indicada por la clave y cambia su valor. Si la celda no existe, lo indica por pantalla.
- `existeClave(String clave)`: devuelve verdadero o falso dependiendo de si existe o no la celda que indica la clave que se pasa en el argumento.
- `existeValor(String valor)`: devuelve verdadero o falso dependiendo de si existe o no alguna celda con el valor que se pasa en el argumento.
- `primerValor(String valor)`: devuelve la clave de la primera celda que tenga el valor pasado como parámetro. Si no existe ninguna clave, devolverá la cadena `"undefined"`.
- `mostrar(diccionario)`: muestra el diccionario con sus pares en columna y ordenados por clave.

```
Clave: ……  Valor: ……
Clave: ……  Valor: ……
Clave: ……  Valor: ……
Clave: ……  Valor: ……
```

132.- Crea una clase para almacenar la ciudad de nacimiento y el DNI de cada alumno/a de un instituto. *IMPORTANTE: la clase almacenará solo Strings. No hay que crear la clase Alumno (ni ninguna otra salvo la principal) en este ejercicio*.

Crea las variables de clase que necesites (todas privadas) e implementa los siguientes métodos:

- `aniadirAlumno(String dni, String ciudad)`: almacena el par nombre-ciudad en la estructura. Si existe ya un alumno con ese nombre, se indica por pantalla y no se almacena nada.
- `mostrarCiudad(String ciudad)`: Si se encuentra en la estructura, muestra la ciudad que se le pasa como parámetro y el número de alumnos que han nacido en ella.
- `cantidadCiudades()`: devuelve el número de ciudades distintas que hay en la estructura.

- `mostrarDatos()`: devuelve una cadena compuesta de cada alumno y su ciudad de nacimiento.
- `borrar(String dni)`: si existe, elimina al alumno con ese DNI. Si no es así, indica un mensaje por pantalla.
- `borrar()`: elimina todos los alumnos.
- `resumen()`: devuelve una cadena donde se indica cada ciudad almacenada y el número de alumnos/as que han nacido en ella.

Ejemplo:

```
Málaga: 23 alumnos.
Granada: 12 alumnos.
Jaén: 5 alumnos.
Madrid: 10 alumnos.
...
```

Crea un fichero **main** para probar la clase anterior. Para ello, muestra un menú para poder realizar las acciones:

1. Añadir Alumno.
2. Borrar Alumno.
3. Mostrar Instituto.
4. Alumnos por ciudad.
5. Mostrar datos. *(Esta opción debe mostrar el numero de ciudades distintas que hay y el numero de alumnos por ciudad)*
6. Borrar todo.
7. Salir

133.- Realiza cada una de las tres partes que se indican a continuación:

PARTE 1

Crea la clase **Empleado**. De un empleado necesitamos su nombre, su apellido (uno solo), sexo, el sueldo base que tiene y los años de antigüedad en al empresa. Todos privados.
La clase tendrá un constructor con el nombre, el apellido y el sexo. El sueldo base será de 800€ y los años de antigüedad será un número aleatrio entre 0 y 10.

Además de los getters y setters que necesites para resolver la segunda parte de este ejercicio, la clase tendrá los métodos: `toString` y `equals` (tú decides cuando un empleado es igual a otro).

PARTE 2:

Haciendo uso de la clase empleado del párrafo anterior, crea la clase **Departamento**, la cual se compondrá de varios empleados almacenados en una estructura según su DNI (tipo Integer). *Nota: la clase empleado no almacena el DNI, por tanto NO PUEDES definir el DNI en dicha clase.*

También son necesarios los siguientes métodos:

- `AniadirEmpleado`: recibe como argumento un DNI y un empleado y lo añade a la estructura.
- `BorrarEmpleado`: recibe como argumento un DNI y ,en el caso de que exista un empleado con ese DNI, la borra.
- `BuscarEmpleado`: recibe como argumento un DNI y muestra los datos del empleado con ese DNI. O un mensaje si no existe un empleado con ese DNI.
- `RevisarSueldos`: este método recorre toda la estructura y actualiza los sueldos según el número de años de la empresa:
  - Entre 0 y 1, no toca el sueldo.
  - Entre 2 y 4, aumenta 200€.
  - Entre 6 y 8, aumenta 500€.
  - Entre 9 y 10, aumenta 750€.
  - Más de 10, aumenta 900€.
- `NuevoAnio`: recorre la estructura y suma un año a todos los empleados.
- `MostrarDepartamento`: muestra todos los datos de cada empleado de manera ordenada.

PARTE 3:

Crea un fichero **main** para probar la clase anterior. Para ello, muestra el siguiente menú:

1. Mostrar Departamento.
2. Añadir Empleado.
3. Despedir Empleado.
4. Aumentar antigüedad al departamento.
5. Revisar salarios.
6. Despedir a todos los empleados.
7. Salir

Programa la funcionalidad de cada entrada del menú usando la clase Empleado.

Lectura y Escritura de ficheros con COLECCIONES

134.- *(Otra versión del ejercicio 130 de la relación de dicccionarios)* Dado el fichero `EJ130.txt`, el cual contiene un texto de varias líneas, implementa un programa que cuente el numero de apariciones de cada palabra de ese fichero.
Para ello crea una estructura que almacene cada palabra y el número de veces que aparece en el texto.

Recuerda que un texto puede tener entre 0 y N palabras (siendo N un número desconocido) y que una palabra puede estar escrita en mayúscula, minúscula o combinación de ambas (y sigue siendo la misma palabra)

La parte final del programa mostrará todas las palabras del texto una debajo de otra junto con el número de veces que aparece cada una.

A tener en cuenta:

- No importa el orden en el que se almacenen/muestren las palabras del texto.
- Vamos a suponer que no hay signos de puntuación en el texto. Sólo letras, números y espacios en blanco.

135.- Modifica el ejercicio 132 de la relación de Diccionarios para que, al ejecutar el programa lea y cargue en memoria (en una estructura) los datos de un fichero de texto llamado `datos.dat` con el siguiente formato: `dni: ciudad`

Pej;

```
76899456A: Jaen
71234567H: Cadiz
55678679K: Malaga
74901349J: Jaen
```

A continuación mostrará el menú con dos nuevas opciones:

1. Guardar instituto: pasa toda la información almacena en la estructura al fichero `datos.dat`
2. Cargar instituto: carga los datos del fichero `datos.dat` y machaca con esos valores la estructura.

Además, al salir del programa SIEMPRE se va a guardar el contenido de la estructura en el fichero `datos.dat`

Importante: El resto de opciones siguen funcionando normalmente. Se guardarán/borrarán alumnos en la estructura, no en el fichero. El fichero sólo se actualizará cuando se elija la opción de Guardar o se salga del programa.