Tema 04 - Arrays, Strings, Algoritmos de Ordenación, HashMaps

Objetivos del Tema:

* Introducir estructuras de datos nuevas, más avanzas que los tipos de datos básicos
* Resolver problemas usando arrays y strings
* Introducir el concepto de la estructura de datos *diccionario* y su implementación mediante una *hashtable.*
* Comprender como funcionan los algoritmos de ordenación

1.- Arrays

Hasta ahora en los problemas que hemos resuelto, cuando ha hecho falta almacenar información de varios objetos hemos declarado tantas variables como objetos necesitábamos, pero imaginemos que nos piden resolver un problema tal como que hay que calcular la altura y edad media de los 30 alumnos de una clase, declarar 30 variables para la altura de cada alumno y otras tantas para la edad, no es una solución muy elegante que digamos, es aquí donde entran en juego los *arrays, vectores o matrices*.

Un *array* no es más que una variable que almacena dentro un conjunto de variables del mismo tipo, a las cuales se accede mediante un índice. En el ejemplo de que hablábamos en al párrafo anterior podrías declarar dos arrays de tipo *int* para almacenar las 30 alturas y las 30 edades, de la siguiente forma:

```
int [] edades=new int[30];
int alturas[]=new int[30];
```

Obsérverse que se inicializan ambas variables *edades* y *alturas* llamando a new. Lo cual quiere decir que en ***JAVA*** los arrays no son otra cosa que objetos.

Para acceder a un valor en particular dentro de un array se usa el operador [] (indexación). En los ejemplos anteriores *edades[0]* o *alturas[0]*, harían referencia al primer valor de tipo *int* almacenado en ambos arrays, y *edades[29]* y *alturas[29]* harían referencia al último valor almacenado en ambos.

Los valores válidos de los índices de un *array* están comprendidos en el rango ***[0, a.length-1]***. El *atributo* ***a.length*** de un array nos da el número de elementos del mismo. En el caso de que intentemos acceder a un índice fuera de rango se producirá una ***java.lang.ArrayIndexOutOfBoundsException***.

2.- Formas de inicializar un *Array*

Ya hemos visto en el punto anterior una forma de inicializar un array:

```
double [] coeficientes=new double[3];
```

También podemos inicializar un *array* con un *literal de array:*

```
double coeficientes[]={1.5,-3.3,7.8};
```

Incluso podríamos utilizar una variable, como en el siguiente ejemplo:

```
java.util.Scanner teclado=new java.util.Scanner(System.in);
System.out.println("Introduce la cantidad de valores que vas a introducir: ");
int n=teclado.nextInt();
int valores[]=new int[n];
```

En este ejemplo se observa que el array se construye *dinámicamente*, es decir no sabemos a priori el número de elementos que va a contener, es el usuario el que nos lo dice a través del teclado.

3.- Propiedades de los *Arrays*

Si los arrays son objetos y en *JAVA* todo objeto desciende de la clase ***java.lang.Object***, esto quiere decir que los Arrays heredan todas las propiedades (atributos y métodos) de la clase *Object*.

- Métodos

equals, no nos sirve de mucho pues lo que compara no son los objetos en si, sino si las referencias *"apuntan"* al mismo objeto (hay objetos que reescriben este método para que si sea de utilidad, por ejemplo, la clase String).

clone, este si que nos va a ser útil. En el caso de los *Arrays* duplica el array.

toString, simplemente imprime el *identificador* del objeto

- Atributos

length nos indica el número de elementos que tiene el *Array*.

Para ***asignar*** un array a otro array no podemos hacelo directamente con el operador de asignación (=), sino que debemos llamar al método ***clone*** del array que queremos asignar, tal y como se ve en el siguiente ejemplo:

```
	int a[]=new int[10];
	int b[];
	b=a.clone();
```

4.- Recorrido de un *Array*

Ejemplo: pongamos que tenemos una clase y queremos calcular la edad media de los alumnos que la componen, el siguiente código resolvería el problema:

```
int numAlumnos;
java.util.Scanner teclado=new java.util.Scanner(System.in);
System.out.println("Introduce el número total de alumnos que tiene la clase: ");
numAlumnos=teclado.nextInt();
int []edades=new int[numAlumnos];
for(int i=0;i<edades.length;i++) {
	System.out.print("Introduce la edad del alumno "+i+": ");
	edades[i]=teclado.nextInt();
}
//una vez leídas todas las edades, procedemos a calcular la media
double media=0.0;
for(int i=0;i<edades.length;i++) {
	media+=edades[i];
}
media/=edades.length;
System.out.printf("La media de los %d alumnos de la clase: es: %.2f\n",edades.length,media);
```

5.- Arrays de múltiples dimensiones

Los arrays multidimensionales en realidad son arrays de arrays de arrays....
Por ejemplo, el array bidimensional
| 1 2 3 |
| 4 5 6 |
| 7 8 9 |
Se declararía de la siguiente forma:

```
int [][] a= {{1,2,3},{4,5,6},{7,8,9}};
```

Para recorrer un array bidimensional haría falta un doble bucle recorriendo filas y columnas. Como en el ejemplo del ejercicio siguiente:

Realizar ejercicio 2 del boletín

6.- Strings

Las cadenas de caracteres en JAVA son objetos de la clase String. En otros lenguajes, como lenguaje ***C*** no existe el tipo de datos *cadena de caracteres*, en su lugar se utiliza un array de caracteres.

* Creación de objetos de la clase ***String*** en JAVA:

- ***String s1="pepito";***
- ***String s2=new String("pepito");***
- ***String s3=new String(s2); //"pepito"***

* Métodos de la clase String:

- ***int length()***, devuelve el número de caracteres de la cadena. Ejemplo: "pepe".length() devolvería 4.
- ***String concat(String s2)***. Devuelve la concatenación del objeto *String* con que se llama con el objeto *String* ***s2***. Ejemplo: String s1=new String("hola"); String s2=", qué tal?"; s1.concat(s2); // devolvería "hola, qué tal?"
  En este caso es preferible utilizar el operador + (sobrecargado), que concatena una cadena con la conversión de lo que esté al otro lado del operador a cadena.
- ***int compareTo(String s)***. Devuelve:
  - <0 Si es menor el String del objeto desde el que se hace la llamada que ***s***
  - 0 Si ambas cadenas son iguales
  - >0 Si es menor ***s*** que el String del objeto desde el que se hace la llamada

  Las mayúsculas están antes en orden alfabético que las minúsculas.
- ***boolean equals(String s)***. Devuelve ***true*** o ***false***, según las cadenas sean o no iguales, léxicograficamente hablando. Distingue mayúsculas de minúsculas
- ***boolean equalsIgnoreCase(String s)***. Devuelve ***true*** o ***false***, según las cadenas sean o no iguales, léxicograficamente hablando. Distingue mayúsculas de minúsculas

  ** **No se debe utilizar el operador == para comparar cadenas.**
- ***String trim()***. Devuelve una cadena igual a la del objeto desde el que se llama al método pero eliminando los espacios del principio y del final.
- ***String toLowerCase()***. Devuelve una cadena con todos los caracteres de la cadena del objeto desde el que se llama convertidos a minúsculas.
- ***String toUpperCase()***. Ídem pero a mayúsculas.
- ***String replace(char c, char nuevoc)***. Reemplaza todas las ocurrencias de ***c*** por ***nuevoc*** en la cadena del objeto desde el que se llama, está sobrecargado y se pueden utilizar también cadenas. P.ej.: *"hola, me voy".replace("hola","adiós")*, daría como resultado *"adiós, me voy"*.
- ***String substring(int ini, int fin)***. Devuelve la cadena de caracteres comprendida entre el carácter que ocupa la posición *ini* y la posición anterior a *fin* de la cadena del objeto desde que se llama. P.ej.: *"hola qué tal".substring(5,8)* devolvería *"qué"*.
- ***boolean startsWith(String s)***. Devuelve true si la cadena del objeto desde el que se llama empieza por la cadena ***s***. P.ej.: *"hola qué tal".startsWith("hola")* devuelve *true*.

- ***boolean endsWith(String s)***. Ídem si finaliza con la cadena ***s***.
- ***char charAt(int i)***. Devuelve el carácter que ocupa la posición ***i*** en la cadena del objeto desde el que se llama. P.ej.: *"hola".charAt(2) ==> 'l'*.
- ***int indexOf(int c)/int indexOf(String s)***. Devuelve la posición de un del (carácter ***c*** | cadena ***s***) dentro de la cadena del objeto desde el que se llama. P.ej: "hola".indexOf('l') ==>2, "hola".indexOf("la") ==> 2
- ***char[] toCharArray()*** devuelve un array con los caracteres que forma el objeto ***String***.
  P.Ej.: *"Pepe".toCharArray()* ==> {'P','e','p','e'}
- ***String valueOf(int i)*** devuelve un ***String*** con el número ***i*** convertido a *String*.
  Ejemplo: *String.valueOf(1234)* ==> "1234"
- ***public boolean matches(String regex)*** devuelve true si la cadena del objeto desde el que se llama ***"encaja"*** en la expresión regular ***regex***. Ejemplo: "172.124.117.100".matches("([0-9]{1,3}\\.){3}[0-9]{1,3}") ==> true
- ***String repeat(int n)*** devuelve un ***String*** con la cadena del objeto desde donde se llama repetida n veces. Ejemplo: "*hola ".repeat(5)* ==> "hola hola hola hola hola "
- ***String.format* método static** que devuelve la cadena que imprimiría ***System.out.printf***.
  Ejemplo: *String.format("Hola me llamo %s y tengo %d años y %5.2f Euros","Juan",33,55.323) ==> "Hola me llamo Juan y tengo 33 años y 55,32 Euros"*
- ***String[] split(String regex, int limit)*** devuelve un array con la cadena trozeada según el separador que le pasemos como parámetro
  separador es una expresión regular
  limit es el tamaño máximo del array de salida
  Ejemplos: "Hola que tal".split(" ") ==> {"Hola", "que", "tal"}

StringTokenizer (esta clase está en desuso debido al método *split*)

Ejemplo:

```
StringTokenizer st=new StringTokenizer("uno dos tres cuatro", " ");
while(st.hasMoreTokens()) {
   System.out.println(st.nextToken());
}
```

Imprimiría:
uno
dos
tres
cuatro

7.- Arrays de objetos

```
String[] s=new String[6]; s ==> String[6] { null, null, null, null, null, null}
```

```
String[] saludos={"hola","hello","hallo","salut"};
saludos ==> String[4] { "hola", "hello", "hallo", "salut" }
```

```
Double[] d=new Double[10]; d ==> Double[10] { null, null, null, null, null, null,
null, null, null, null }
for(int i=0;i<10;i++) { d[i]=new Double(i); }
d ==> Double[10] { 0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0 }
d[0].toString() ==> "0.0"
d[5].toString() ==> "5.0"
```

```
public class Persona {
    public String nombre;
    public int edad;
    public Persona(String nombre,int edad) {
        this.nombre=nombre;
        this.edad=edad;
    }
    public int getEdad() {
        return this.edad;
    }
}
  created class Persona
Persona[] personas=new Persona[5]; personas ==> Persona[5] { null, null, null,
null, null }
personas[0]=new Persona("Juan",33);
personas[0].nombre ==> "Juan"
personas[0].edad ==> 33
personas[1].nombre
|  Exception java.lang.NullPointerException
personas[1]=new Persona("Luisa",25)
personas[1].nombre ==> "Luisa"
personas[1].edad ==> 25
personas ==> Persona[5] { Persona@23e028a9, Persona@cd2dae5, null, null, null }
personas[0] ==> Persona@23e028a9
personas[1] ==> Persona@cd2dae5
personas[2] ==> null
```

8.- Algoritmos de ordenación

- ***Burbuja***

```
    public static void swap(int[] v, int i, int j) {
      int aux=v[i];
      v[i]=v[j];
      v[j]=aux;
    }
    public static void burbuja(int[] vector) {
        int i, j, aux;
        for (i = 0; i < vector.length - 1; i++) {
            for (j = 0; j < vector.length - i - 1; j++) {
                if (vector[j] > vector[j + 1]) {
                  swap(vector,j,j+1);
                }
            }
        }
    }
```

Ejemplo de ejecución:

50	26	7	9	15	27	Array original
Primera pasada:						
26	**50**	7	9	15	27	Se intercambian el 50 y el 26
26	7	**50**	9	15	27	Se intercambian el 50 y el 7
26	7	9	**50**	15	27	Se intercambian el 50 y el 9
26	7	9	15	**50**	27	Se intercambian el 50 y el 15
26	7	9	15	27	**50**	Se intercambian el 50 y el 27
Segunda pasada:						
7	**26**	9	15	27	50	Se intercambian el 26 y el 7
7	9	**26**	15	27	50	Se intercambian el 26 y el 9
7	9	15	**26**	27	50	Se intercambian el 26 y el 15

Ya están ordenados, pero los dos bucles for seguirán ejecutándose hasta el final. El tiempo de ejecución del algoritmo de la burbuja es del orden $O(n^2)$. Es uno de los peores algoritmos de ordenación en cuanto a tiempo de ejecución, solamente es recomendable su uso para ordenar listas con un número pequeño de elementos.

https://www.youtube.com/watch?v=EQMGabLO_M0
https://www.youtube.com/watch?v=93fHPCi1mC8
https://www.youtube.com/watch?v=6Gv8vg0kcHc

- Selection Sort

```
    public static void seleccion(int[] vector) {
      for(int i=0;i<vector.length-1;i++) {
            for(int j=i+1;j<vector.length;j++) {
                  if (vector[i]<vector[j]) {
                        swap(vector,i,j);
                  }
            }
      }
    }
```

https://www.youtube.com/watch?v=cqh8nQwuKNE
https://www.youtube.com/watch?v=Ns4TPTC8whw
https://www.youtube.com/watch?v=ZMO3Fow05tg

9.- Búsqueda dicotómica o binaria en *Arrays ordenados*

```
    public static int busquedaBin(int[] ordenado, int b) {
        int posInf=0, posSup=ordenado.length-1, centro;
        while(posInf<=posSup) {
            centro=(posSup+posInf)/2;
            if (b==ordenado[centro]) {
                return centro;
            } else if (b<ordenado[centro]) {
                posSup=centro-1;
            } else {
                posInf=centro+1;
            }
        }
        return -1;
    }
```

10.- La clase *HashMap*

https://drive.google.com/open?id=1RBP4RsjCJPcj_qkwk5o4JQ4Te196dGrC

```
jshell> HashMap h=new HashMap()
h ==> {}
jshell> h.put("Juan",23)
|  Warning:
|  unchecked call to put(K,V) as a member of the raw type java.util.HashMap
|  h.put("Juan",23)
|  ^--------------^

jshell> h.put("Luisa",21)
|  Warning:
|  unchecked call to put(K,V) as a member of the raw type java.util.HashMap
|  h.put("Luisa",21)
|  ^---------------^

jshell> h.put("Alicia",26)
|  Warning:
|  unchecked call to put(K,V) as a member of the raw type java.util.HashMap
|  h.put("Alicia",26)
|  ^----------------^

jshell> h.put("Francisco",24)
|  Warning:
|  unchecked call to put(K,V) as a member of the raw type java.util.HashMap
|  h.put("Francisco",24)
|  ^-------------------^

jshell> System.out.println(h.toString())
{Francisco=24, Alicia=26, Juan=23, Luisa=21}
```

```
jshell> HashMap<String,Integer> h=new HashMap<String,Integer>();
h ==> {}
jshell> h.put("Juan",23)
jshell> h.put("Luisa",21)
jshell> h.put("Alicia",26)
jshell> h.put("Francisco",24)
jshell> System.out.println(h.toString())
{Francisco=24, Alicia=26, Juan=23, Luisa=21}
jshell> h.put(3,3)
|  Error:
|  incompatible types: int cannot be converted to java.lang.String
|  h.put(3,3)
|        ^
```

Boletín de Problemas

Arrays

1.- Reimplementar la clase ***vectorTridimensional*** del tema anterior, como ***Vector*** donde un vector puede tener ***n dimensiones***. Se deben implementar la suma, resta, producto escalar, producto vectorial y módulo del vector, y un método numVectores que nos devuelva el número de objetos Vector creados en nuestra aplicación. La clase debe tener 3 constructores uno que reciba un número entero (dimensión del vector), otro que reciba un array de números *double* y un *constructor de copia*.
Ejemplos de utilización de la clase:
- *Vector v1=new Vector(10), v2=new Vector(v1), v3=new Vector()*
- *Vector v3=v1.suma(v2)*
- *double modulo=v3.modulo()*
- *v3=v1.resta(v2);*
- *double pe=v1.prodEscalar(v2)*
- *v3=v1.prodVectorial(v2);*

2.- Implementar una clase M***atrizBid*** que represente una matriz bidimensional y que implemente las operaciones de ***suma***, ***resta*** y ***producto***, añadir un método *getNumMatrices* que devuelve el número de objetos de la clase ***MatrizBid*** que se han creado.

3.- Para resolver los ejercicios siguientes crearemos una clase de *utilidad* llamada ***UtilidadesMatrices*** en la cual todos los métodos serán *static*. Por ejemplo, el ejercicio siguiente quedaría encuadrado dentro de la clase, de la siguiente forma:

```
public class UtilidadesMatrices {
    public static int[] suma(int[] a, int []b) {
      .........
    }
}
```

4.- Añadir a la clase del ejercicio anterior el método ***public int[] suma(int a[], int b[])***, el cual recibe dos *arrays*, y devuelve un array que representa la suma de todos los elementos que ocupan la misma posición en ***a*** y ***b***.

5.- Añadir a la clase el método *arrayBidSuma* que recibe dos arrays bidimensionales de números enteros y devuelve un array bidimensional con la suma de los elementos de la misma posición de ambos.

6.- Añadir a la clase un método *construyeArray* que reciba un número entero y devuelva un array de números enteros con sus elementos inicializados a la posición de cada elemento (sólo que comenzando por 1 en lugar de 0). Por ejemplo, si se le pasa como parámetro ***5***, devolvería el array de números enteros: {1,2,3,4,5}.

7.- Añadir a la clase un método *tablaMultiplicar* que devuelve un array con la tabla de multiplicar del número que se le pase por parámetro.

8.- Añadir a la clase un método *maximo* que reciba un array de enteros y nos devuelve el mayor.

9.- Ídem *minimo*

10. Ídem mínimo y máximo a la vez (array de dos elementos donde el 1º es el mínimo y el 2º el max.)

11. Método *posicionArray* que recibe un array de enteros y un número entero y me devuelve la posición de éste último dentro del array (-1 en caso de que no exista)

11.2 - Utilizando el método anterior crear un nuevo método posicionesArray que reciba un array de enteros y un número y nos devuelva un array con las posiciones donde aparece dicho número (array vacío en caso de que no exista).

Ejemplos: posicionesArray({2,3,4,1,2,5,7,9,5},5) ==> {5,8}
posicionesArray({2,3,4,1,2,5,7,9,5},9) ==> {7}

12.- Añadir un método *mediaArray* que reciba un array de enteros y devuelva su media.

13.- Añadir un método *invierteArray* que devuelva el array de números enteros pasado como parámetro en orden inverso.

14.- Añadir un método *arrayAleatorio* que devolverá un array de ***n*** números enteros aleatorios comprendidos entre ***inf*** y ***sup*** (ambos inclusive) y que no tenga ningún elemento repetido. P.ej.: arrayAleatorio(6,1,49) ==> {1,7,6,5,49,39}

15.- Lo mismo del ejercicio 5, sólo que cada posición contiene la suma de las posiciones anteriores. Por ejemplo si se la pasa como parámetro ***6***, devolvería el array {1,1,2,4,8}.

16.- La criba de Eratóstenes, renombrado astrónomo y geógrafo griego del s. III a. de C. (como todos sabéis), es una técnica para generar números primos. Se comienza escribiendo todos los enteros impares desde 3 hasta N; a continuación se elimina cada tercer elemento después de 3,

cada quinto elemento después de 5, etc., hasta que los múltiplos de todos los enteros impares menores que raíz cuadrada de N hayan sido eliminados. Los enteros que quedan constituyen la lista exacta de los números primos entre 3 y N. Añadir el método cribaEratostenes que devuelva una matriz de booleanos que nos indica si el número de la posición correspondiente del índice es primo (true). Por ejemplo una llamada al método cribaEratostenes(13) debería devolver la matriz {false,true,true,true,false,true,false,true,false,false,false,true,false,true}
[0] [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11] [12] [13]

17.- Añadir un método *eliminaRepetidos* que reciba una matriz de enteros y devuelva una matriz con los elementos repetidos eliminados. P.ej.: *eliminaRepetidos(new int[]{4,3,2,1,4,2,1,5,7,5})* devolvería: {4,3,2,1,5,7}

Cadenas

0.- Crear una clase de *utilidad* UtilidadesStrings a la que iremos añadiendo los métodos *static* que se pieden en los siguientes ejercicios:

1.- Método ***cuentaRepeticiones*** que recibe un carácter y una cadena y nos devuelve el número de veces que el carácter aparece dentro de la cadena.

2.- Método ***cuentaRepeticiones*** que recibe dos cadenas y nos dice cuantas veces aparece la primera dentro de la segunda

3.- Método ***quitaEspacios*** que recibe una cadena y nos devuelve dicha cadena con todos los espacios eliminados

4.- Método ***invertir*** que recibe una cadena s y nos devuelve la cadena "escrita al revés"

5.- Método **palindromo** que recibe una cadena s y devuelve true si s es palíndromo

6.- Método ***toUpper*** que convierta una cadena todo a mayúsculas. P.ej.: *toUpper("Juan") -- > "JUAN"*

7.- Método ***toLower*** que convierta una cadena todo a mayúsculas. P.ej.: *toLower("JuaN") -- > "juan"*

*8.- Método **ascii** que escriba la tabla de códigos ascci de A-Z a-z 0-9*

Extra Arrays

1.- Una fábrica de automóviles produce cuatro modelos de coches cuyos precios de venta son respectivamente 1.5, 1.75, 2.42, 2.6 millones de pesetas. Esta empresa cuenta con cuatro centros de distribución y venta. Se dispone de una relación de datos correspondientes al tipo de vehículo vendido y punto de distribución en que se produjo la venta del mismo. Realizar un programa que leyendo del teclado la relación de datos anterior (100 como máximo), calcule e imprima:

a) Volumen de ventas total.

b) Volumen de ventas por centro.

c) Porcentaje de unidades totales vendidas en cada centro.

d) *Porcentaje de unidades de cada modelo vendidos en cada centro, sobre el total de ventas de la empresa.*

Ejemplo de datos de entrada:

0 1 // modelo de coche 0, punto de venta 1

1 2 // modelo de coche 1, punto de venta 2

2.- Método que devuelva los dígitos de control de una cuenta bancaria (ver notas al final de este documento)

3.- Ver el vídeo https://www.youtube.com/watch?v=t9OW-ouW2RE sobre la implementación del método de ***ordenación de la burbuja*** en donde se implementa el mismo en lenguaje ***C***, adapta el método implementado en ***JAVA*** más arriba en estos apuntes a la implementación que se ve en el vídeo en el cual se introduce una optimización: la introducción de una variable *centinela* ***ordenado,*** en nuestro caso será una variable *boolean* en lugar de una variable tipo *int* ya que en lenguaje ***C*** no existe el tipo de datos *boolean*. Con el uso de este variable se detecta si no se han intercambiado elementos, en cuyo caso quiere decir que el ***array*** (vector) ya está ordenado y no hace falta seguir dando más pasadas. Otra diferencia entre lenguaje ***C*** y ***JAVA*** es que en lenguaje ***C*** los *arrays* no guardan su tamaño, por eso verás que las funciones tienen que recibir además del *array* un segundo parámetro de tipo *int* con el tamaño del array, en *JAVA* disponemos de la propiedad *a.length* que guarda el tamaño del array, ya que los arrays son objetos a diferencia de lo que ocurre en lenguaje ***C*** que no es un lenguaje orientado a objetos. Resuelve también en *JAVA* el problema planteado en el vídeo de leer las notas de 20 alumnos y obtener la media de los tres alumnos que mejor nota han obtenido. Utilizar un fichero de texto para la entrada igual que se hizo en ejercicios anteriores.

4.- Método que reciba dos arrays de números enteros ordenados y devuelve un único array con todos los elementos de los dos arrays ordenados.

5.- Ver el vídeo https://youtu.be/0_PO-77gu_E. Diseñar una matriz bidimensional de 12 filas (meses del año 2019) y un número de columnas de 31,28,31,30,31,30,31,31,30,31,30,31 columnas respectivamente, donde las filas representen los meses del año y las columnas la temperatura media de cada día. Para hacer una prueba podemos guardar los siguientes datos en un fichero de texto y leerlo como ya se hizo en ejercicios anteriores:

temperatuas	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]	[16]	[17]	[18]	[19]	[20]	[21]	[22]	[23]	[24]	[25]	[26]	[27]	[28]	[29]	[30]
[0]	5	5	11	9	8	12	12	6	12	9	6	12	8	10	14	8	10	9	14	7	8	13	5	12	14	8	6	8	13	10	7
[1]	12	8	9	14	6	13	7	9	7	12	8	7	10	8	7	9	7	7	11	7	9	7	13	8	11	10	8	14			
[2]	13	17	17	17	18	7	8	11	7	11	11	6	15	5	14	14	6	14	13	5	12	17	5	13	10	17	11	8	5	7	14
[3]	8	10	5	5	7	17	19	19	14	9	19	5	9	17	11	6	14	5	9	21	9	8	10	6	6	15	17	19	10	12	
[4]	6	14	19	20	18	16	10	10	7	22	17	20	23	19	19	20	11	9	10	23	10	10	17	23	22	18	6	17	10	11	20
[5]	26	15	25	17	17	18	12	9	13	5	16	26	23	6	24	13	11	6	19	25	18	17	8	20	10	27	7	25	19	25	
[6]	14	10	9	19	34	7	13	14	32	12	25	26	8	23	28	5	10	14	17	26	33	14	34	23	15	20	5	27	6	31	26
[7]	25	30	20	8	6	8	36	30	32	24	18	6	14	25	9	38	8	35	33	36	22	14	38	25	15	39	36	36	13	10	8
[8]	11	21	8	12	22	19	20	32	18	9	17	34	13	18	12	15	24	34	6	24	7	16	24	16	22	34	24	16	29	18	
[9]	18	5	23	29	10	7	22	6	16	5	9	21	16	12	10	13	12	28	18	11	24	25	20	5	21	17	19	26	11	28	8
[10]	9	17	17	10	17	5	18	7	18	8	19	9	6	12	19	6	15	5	9	14	7	16	11	14	18	13	18	14	8	12	
[11]	10	10	7	9	7	8	13	9	10	10	14	13	11	11	5	13	9	11	9	7	6	14	13	9	11	3	4	12	7	7	14

Puedes descargar el fichero de texto con los datos de la plataforma

Calcular:

a) La temperatura media de cada mes.

b) El día más caluroso y el más frío de cada mes.

c) El día más caluroso del año.

d) El día más frío del año

6.- Realizar un algoritmo que lea una matriz cuadrada de dimensión máxima N*N y haga lo siguiente:

a) Imprimir la media de los elementos de la diagonal principal.

b) Imprimir la media de los elementos de la diagonal secundaria.

c) Imprimir la suma de los elementos por debajo de la diagonal principal y por encima de ella.

d) Imprimir la suma de los elementos por debajo de la diagonal secundaria y por encima de ella

7.- Una empresa tiene no más de 100 trabajadores repartidos en 4 categorías profesionales. La empresa dispone de un array de dos dimensiones donde cada posición contiene el número de horas extraordinarias trabajadas por cada uno cada día del mes.

También se dispone de dos arrays, uno con el código de cada trabajador (de 1 a 100) y la categoría correspondiente y otro con las cuatro categorías junto con el precio de las horas extras por categoría. Calcular:

a) Cuánto ha cobrado cada trabajador en ese mes por horas extraordinarias, presentando en pantalla el código de cada empleado junto a lo cobrado.

b) El total pagado por dichas horas.

Se te entregan 3 ficheros con los datos con los códigos de empleado-categoría a la que pertenecen, categoría-precio hora extra y empleado-horas trabajadas en el mes.

HashMaps

1.- Diseña un programa capaz de contar cuántas veces aparece cada palabra en un texto. El texto se leerá desde un fichero cuyo nombre nos proporcionará el usuario desde la línea de comandos del sistema. Las palabras van separadas por carácteres en blanco, comas, puntos, punto y coma, salto de línea y tabuladores. El informe se dará ordenado de mayor a menor frecuencia de aparición.

2.- Una compañía que se dedica a la venta de secadores de pelo para perros tiene N vendedores. Cada operación que hace un vendedor se registra en una ficha que contiene: número del vendedor (NUMVEN), nombre del vendedor (NOMVEN) y monto de la venta (MONVEN). El número de fichas de entrada no se conoce, y el total de ventas realizadas puede variar de un vendedor a otro. Las fichas no están ordenadas. Realizar un programa que imprima, para cada vendedor, una línea que contenga nº de vendedor, nombre y total de ventas realizadas.

La entrada finaliza cuando encontramos un vendedor con nº -1 y deben efectuarse menciones especiales para los vendedores que logren los dos totales más altos.

NOTAS

- Sobre la cadena de formato que recibe ***System.out.printf*** o el método ***static*** de la clase ***String***, ***String.format***: *https://www.journaldev.com/28692/java-printf-method*

- Dígitos de control de una cuenta bancaria

Para el cálculo de los dígitos de control de una cuenta bancaria se emplea un algoritmo en el que se utilizan los restos módulo 11 de las diez primeras potencias de 2, así como los dígitos de la entidad bancaria, la oficina y la cuenta. Si son b1,b2,b3,b4 los dígitos de la entidad bancaria; o1,o2,o3,o4 los cuatro dígitos de la oficina y c1,c2,c3,c4,c5,c6,c7,c8,c9,c10 los diez dígitos de la cuenta el cálculo que se debe realizar es el siguiente:

1. Para el primer dígito de control calculamos:
   suma1=4*b1+8*b2+5*b3+10*b4+9*o1+7*o3+3*o3+6*o4
   El dígito de control es la diferencia entre 11 y el resto de la división de la suma1 anterior entre 11, salvo que la diferencia sea 11, en cuyo caso el dígito de control es 0 o que la diferencia sea 10, en cuyo caso el dígito de control es 1.
2. Para el segundo dígito de control calculamos:
   suma2=1*c1+2*c2+4*c3+8*c4+5*c5+10*c6+9*c7+7*c8+3*c9+6*c10
   Como en el caso anterior, el dígito de control es la diferencia entre 11 y el resto de la división de la suma2 anterior entre 11, salvo que la diferencia sea 11, en cuyo caso el dígito de control es 0 o que la diferencia sea 10, en cuyo caso el dígito de control es 1.