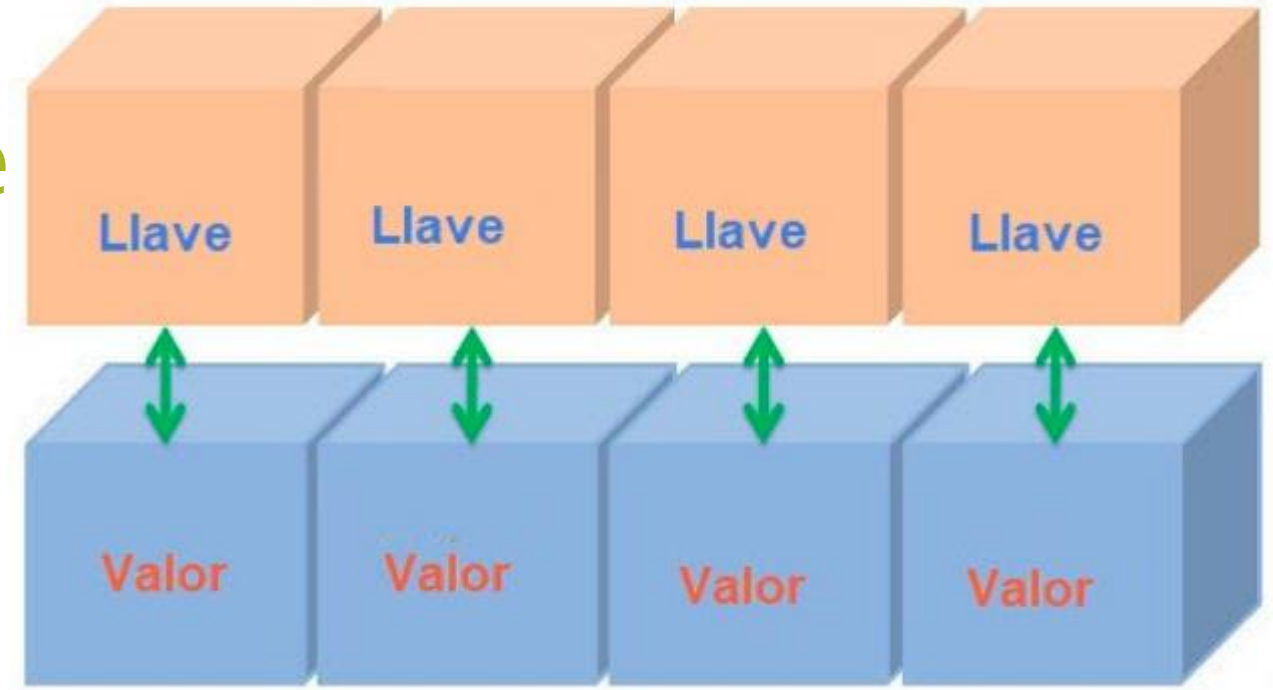
# TEMA 6c: algoritmos de ordenación.




**CICLO GRADO SUPERIOR**

DAM: Diseño Aplicaciones Multiplataforma

DAW: Diseño de Aplicaciones Web

# INDICE

# 6b.1- Definición de Algoritmo.

- Conjunto ordenado de operaciones sistemáticas que permite hacer un cálculo y hallar la solución de un tipo de problemas.
- Algoritmos de ordenación de un array.

| Métodos Iterativos | Métodos recursivos |
|---|---|
| • Burbuja (Bubblesort)<br>• Inserción<br>• Selección<br>• Shellsort | • Ordenamiento por Mezclas (merge)<br>• Ordenamiento Rápido (quickSort) |

# 6b.2- Algoritmos de ordenación.

Existen muchos algoritmos de ordenación para ordenar vectores o matrices.

- El factor más importante es el tiempo de demora junto con la complejidad.
- Nos servirá para aprender a realizar comparaciones. En tiempos de ejecución, ver prerrequisitos, alcance, etc…

Algoritmos más populares:

Bubblesort (método de la burbuja)

QuickSort (método rápido)

# 6b.3- Método de la Burbuja.

El algoritmo más simple de ordenación, aunque el más lento:

6 5 3 1 8 7 2 4

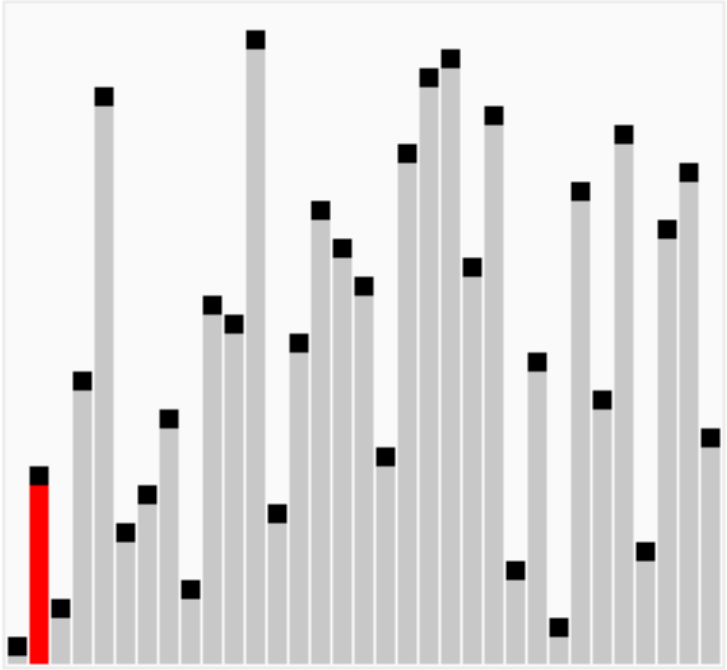
Ordenamiento de burbuja - Wikipedia, la enciclopedia libre

# 6b.3- Método de la Burbuja.

```
for (i=1; i<LIMITE; i++) {
        for (j=0 ; j< LIMITE -1 ; j++){
           if (vector[ j ] >vector[ j+1]) {
                temp = vector[j];
                vector[j] = vector[j+1];
                vector[j+1] = temp;
           } //Fin if
        } //Fin 2º FOR
} //Fin 1er FOR
```

# 6b.4- Método de la Burbuja Mejorada.

- Limita el numero de comparaciones, eliminando las que están sobrando.
- Ordena de forma bidireccional, obteniendo el mayor y el menor.



Ordenamiento de burbuja bidireccional - Wikipedia, la enciclopedia libre

# 6b.5- Método por inserción.

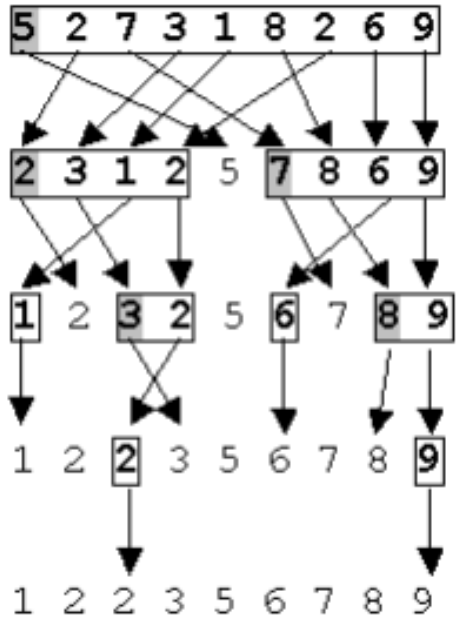
## Coge un elemento e inserta cada uno en el lugar adecuado

```
Insercion(int matrix[]){
        int i, temp, j;
        for (i = 1; i < matrix.length; i++){
          temp = matrix[i];
          j = i - 1;
          while ( (matrix[j] > temp) && (j >= 0) ){
                matrix[j + 1] = matrix[j]; j--;
          }
          matrix[j + 1] = temp;
        }
}
```

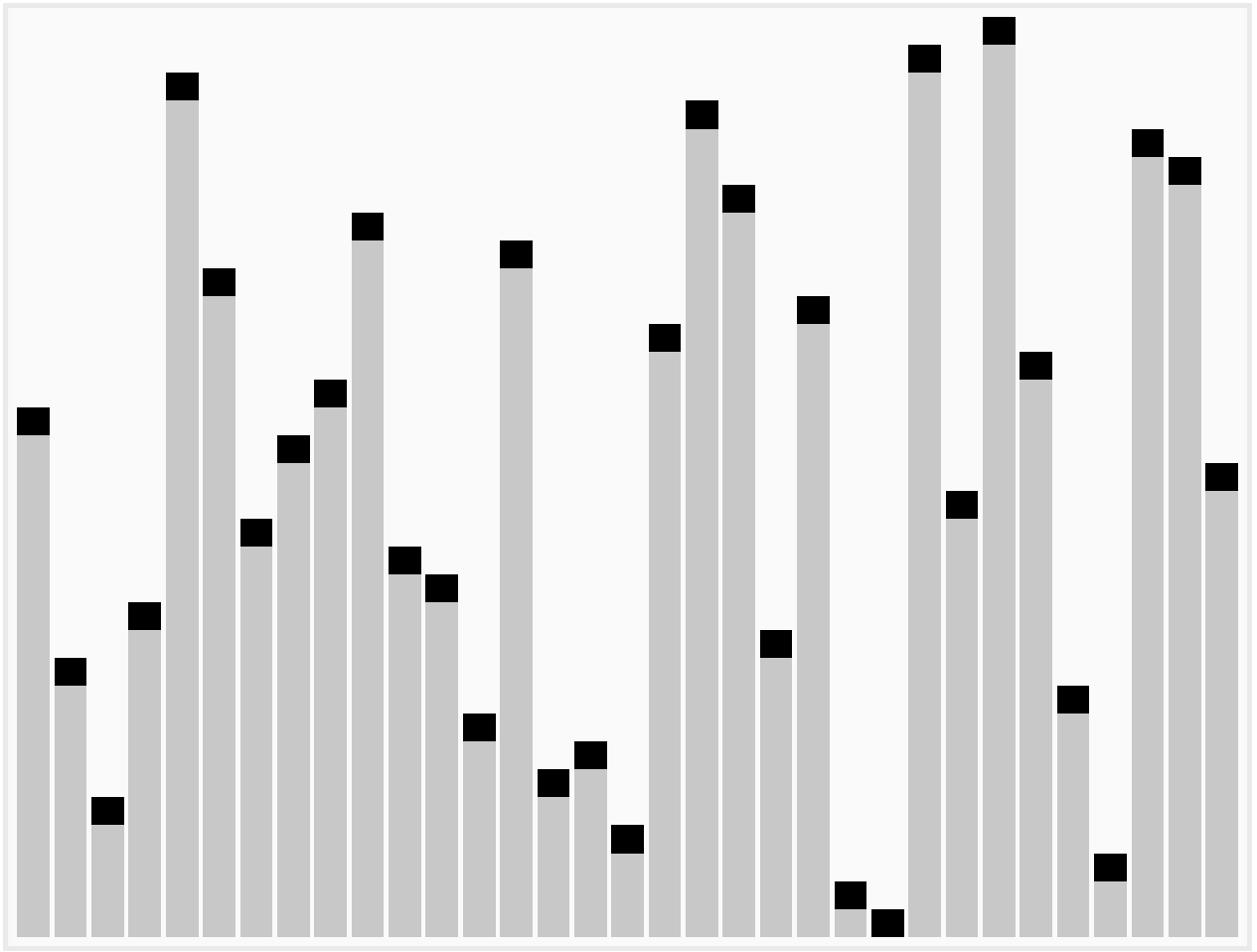
6 5 3 1 8 7 2 4

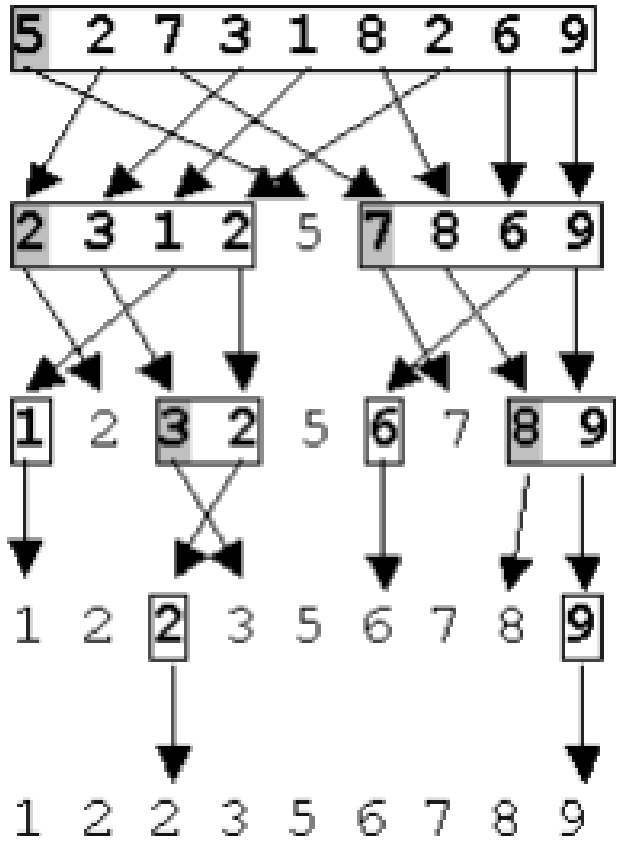
# 6b.6- Método rápido (QuickSort).

- Es el método más rápido mediante ola técnica divide y vencerás.
- Divide el vector en partes iguales, con elemento de inicio, pivote y fin. El pivote es el comodín que nos permitirá segmentar la lista.
- Los valores mayores los deja a la derecha del pivote, y los menores a la izquierda.



# 6b.6- Método rápido (QuickSort).






Quicksort - Wikipedia, la enciclopedia libre

# 6b.7- Comparación de tiempos.

## Coge un elemento e inserta cada uno en el lugar adecuado

| 262144 elementos | | 2097152 elementos | |
|---|---|---|---|
| Burbuja: | 178.205 | Burbuja: | 11978.107 |
| Seleccion: | 158.259 | Seleccion: | 10711.01 |
| Insercion: | 94.461 | Insercion: | 7371.727 |
| Rapido: | 0.061 | Rapido: | 0.596 |
| Shell: | 0.086 | Shell: | 0.853 |
| Merge: | 0.201 | Merge: | 1.327 |

# LIBRERIAS Y MÉTODOS EN JAVA CON ARRAYS

# 6b.7- Librerias de Java.

- Java dispone de librerías propias para ordenar un array.
- Se requiere importar la librería ***java.util.Arrays***.

```
import java.util.Arrays;
public static void main(String[] args) {
    int[] ejemplo05 ={9,5,3,4,1,6};
    Arrays.sort(ejemplo05);
    imprimirArray(ejemplo05);
}
```

```
run:
Valores del Array:[1,3,4,5,6,9]
BUILD SUCCESSFUL (total time: 0 seconds)
```

**Actividad de clase**
Implementar el método imprimir array para obtener el resultado, sin coma al final

# 6b.7- Librerías de Java.

```
import java.util.Arrays;
public static void main(String[] args) {
     String[] nombres = {“Juan”,”Sara”,”Ana”, ”Luis”,
”Pepe”, “Elena”};
     Arrays.sort(nombres);
     imprimirArray(nombres);
}
```

# 6b.7- Librerías de Java.

- Ordenar de forma inversa
- Se requiere importar la librería ***java.util.Collections***.

```
import java.util.Collections;
import java.util.Arrays;
public static void main(String[] args) {
    String[] nombres = {“Juan”,”Sara”,”Ana”, ”Luis”,
”Pepe”, “Elena”};
    Arrays.sort(nombres,Collections.reverseOrder());
    imprimirArray(nombres);
}
```

# 6b.7- Método de Java COPIAR UN ARRAY

- Para copiar un array Java ofrece la librería del sistema para copiar arrays.

```
System.arraycopy(arrayOrigen, Inicio, arrrayDestino,
inicioArrrayDestino, numero de elementos a copiar);
```

- El array destino ha de ser inicializado previamente.
- Pueden producirse excepciones en los siguientes casos:

➔ Copiar fuera del área reservada del array (IndexOutOfBoundsException)

➔ Arrays de diferentes tipos (ArrayStoreException)

# 6b.7- Fusionar Método de Java COPIAR UN ARRAY

- Fusionar un array consiste en unir dos o más arrays, preferentemente del mismo tipo.
- Existen varios métodos y formas de aplicarlo.

a) Crear un array con la longitud de la suma de ambos y concatenarlos.

b) Hacer uso de la interfaz STREAM.

c) Implementarlo.

d) Con un método propio de Java.

# 6b.7- Fusionar Método de Java COPIAR UN ARRAY

**Método I**:

```
tipo [ ]  arrayFusion = new tipo
[array1.length+array2.length];
```

***Aplico ARRAYCOPY:***

```
System.arraycopy(array1, 0, arrayFusion, 0,
array1.length);

System.arraycopy(array2, 0, arrayFusion,
array1.length, array2.length)
```

# 6b.7- Fusionar Método de Java COPIAR UN ARRAY

**Método II**: interfaz STREAM para un tipo de dato String.

```
import java.util.stream.Stream;

String [ ] arrayFusion = Stream.of(array1,
array2).flatMap(Stream::of).toArray(String[
]::new);

System.out.println(Arrays.toString(arrayFusion);
```

Más información:
https://docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html

# 6b.7- Fusionar Método de Java COPIAR UN ARRAY

**Método III**: Realizando un método que recorra los dos arrays y los copie mediante el bucle FOR o WHILE.

Realizarlo en la tarea.

**Método IV:** con ArrayUtils. Se puede aplicar a cualquier tipo de datos

```
String[] concatenados=
(String[])ArrayUtils.addAll(array1, array2);
```

# CLASES y ALGORITMOS EN JAVA

# REPASO MODIFICADORES DE ACCESO.

| MODIFICADOR | CLASE | PACKAGE | SUBCLASE | TODOS |
|---|---|---|---|---|
| `public` | Si | Si | Si | Si |
| `protected` | Si | Si | Si | No |
| `private` | Si | No | No | No |
| `No especificado` | Si | Si | No | No |

Se aplica a todos los miembros de una clase, tanto a métodos como a atributos.

Se considera ***restricitvo*** cuando se limita el acceso a una subclase.

# 6b.8- ARRAYS DE OBJETOS.

- En lenguaje C se dispone de estructuras para almacenar atributos de distintos tipos.
- Java dispone de clases para almacenar estructuras.
  - Se crea la clase con sus atributos.
  - Se implementan los constructores.
  - Se va recorriendo el array para imprimir sus elementos.

Recordar que Java implementa ***ArrayIndexOutOfBoundsException*** si se accede con un índice ilegal al array

# 6b.8- ARRAYS DE OBJETOS.

- Me creo métodos propios para imprimir el array:

```
public String toString() {
…
}
```

- Me creo un comparador de objetos con la clase creada:

```
class compararAtributo implement
    Comparator<clase_a_comparar>{
    ......
    ......
}
```

Para llamarlo: `Arrays.sort(array, objeto_clase_externa.new compararAtributo());`

# 6b.8- ARRAYS DE OBJETOS.

## Comparadores de cadenas:

a) **Método equals().** Devuelve true si son iguales y false si no lo son

```
cadena1.equals(cadena2);
```

**b) Método compareTO,** devuelve 0, -1 y 1 si son iguales, menor o mayor.

```
cadena1.compareTo(cadena2)==0
cadena1.compareTo(cadena2)<0
cadena1.compareTo(cadena2)>0
```

# 6b.8- ARRAYS DE OBJETOS.

## Comparadores de cadenas:

**c) Método collator().** Permite definir criterios como si se tratan o no mayúsculas, minúsculas o letras acentuadas.

**Collator.PRIMARY:** no distingue mayúsculas, minúsculas y letras acentuadas. (A=a=á)

**Collator.SECONDARY:** no distingue mayúsculas, minúsculas pero si letras acentuadas. (A=a!=á)

**Collator.TERTIARY:** distingue mayúsculas, minúsculas y letras acentuadas. (A=a!=á)

Similar al compareTo

# 6b.8.1- MÉTODO Compare To. Ejemplo 01

```
public class Sample_String {
    public static void main(String[] args) {
        String str_Sample = "a";
        System.out.println("Compare To 'a' b is : " + str_Sample.compareTo("b"));
        str_Sample = "b";
        System.out.println("Compare To 'b' a is : " + str_Sample.compareTo("a"));
        str_Sample = "b";
        System.out.println("Compare To 'b' b is : " + str_Sample.compareTo("b"));
    }
}
```

**Salida:**

Compare con ‘a’ b es: -1

Compare con ‘b’ a es: 1

Compare con ‘b’ b es: 0

# 6b.8.2- MÉTODO CompareToIgnoreCase

Aplicar el método para que no sea sensible a mayúsculas y minúsculas.

```
public class Sample_String {
  public static void main(String[] args) { //Compare to a String String
        str_Sample = "RockStar";
        System.out.println("Compare To 'ROCKSTAR': "+str_Sample.compareTo("rockstar"));
        //Compare to - Ignore case
        System.out.println("Compare To 'ROCKSTAR' - Case Ignored: " +
str_Sample.compareToIgnoreCase("ROCKSTAR"));
  }
}
```

**Salida:**

Comparar con ‘ROCKSTAR’: -32

Comparar con ‘ROCKSTAR’ – Caso ignorado: 0

# 6b.8.3- USAR EL MÉTODO compareTo()

Se utiliza para comparar dos cadenas lexicográficamente. Cada carácter se convierte en un valor Unicode.
El resultado puede ser:

- Si esta cadena es menor que el parámetro de cadena, devuelve un valor inferior a cero;

- Si esta cadena es mayor que el parámetro de cadena, se devuelve un valor mayor que 0.

- si a1 == a2, devuelve 0

# 6b.8.3- USAR EL MÉTODO compareTo(). Ejemplo 3

```
public class Compare {
 public static void main(String[] args) {
  String s1 = "Dam1";
  String s2 = "Dam2";
  System.out.println("String 1: " + s1);
  System.out.println("String 2: " + s2);
  // Compare the two strings.
  int S = s1.compareTo(s2);
  // Show the results of the comparison.
  if (S < 0) {
  System.out.println("\"" + s1 + "\"" + " is lexicographically higher than " + "\""
+ s2 + "\"");
  } else if (S == 0) {
  System.out.println("\"" + s1 + "\"" + " is lexicographically equal to " + "\"" +
s2 + "\"");
  } else if (S > 0) {
  System.out.println("\"" + s1 + "\"" + " is lexicographically less than " + "\"" +
s2 + "\"");
  }
 } }
```

# 6b.8.4- Aplicación a un array de objetos.

Hay que realizar los siguientes pasos:

1º) Crear una clase con una interfaz comparable.

2º) Implementar el método **compareTo** con el campo a comparar de la clase.

```
class clase01 implements Comparable<clase01> {
    // resto de código
    public int compareTo(clase01 objeto) {
    ...
    }
```

3º) Aplicar Libreria de ordenación Array.sort()

# 6b.8.4- Aplicación a un array de objetos.

Hay que realizar los siguientes pasos:

1º) Crear una clase con una interfaz comparable.

2º) Implementar el método **compareTo** con el campo a comparar de la clase.

```
class clase01 implements Comparable<clase01> {
    // resto de código
    public int compareTo(clase01 objeto) {
    ...
    }
```

3º) Aplicar Libreria de ordenación Array.sort()

# Datos de tipo ENUM

# 6b.9- Datos ENUM.

- Un tipo enumerado restringe los valores que se pueden tomar, ayudando a reducir errores en el código.
- Sintaxis:

```
Tipo Público/Privado enum nombreTipoEnumerado {
ELEMENTO1, ELEMENTO2, ELEMENTO3, …, ELEMENTOn };
```

- Sus nombres se escriben en letras mayúsculas para recordarnos que son datos fijos.
- No son enteros ni cadenas, por lo que no se pueden comparar.
- Se pueden declarar dentro o fuera de una clase, con sus correspondientes métodos **getter** y **setter**.

# 6b.9- Datos ENUM.

```
enum Color { ROJO, VERDE, AZUL; }
public class Test {
     public static void main(String[] args) {
          Color c1 = Color.ROJO;
          System.out.println(c1);
     }
}
```

# ALGORITMOS DE ORDENACIÓN y ARRAYS

END