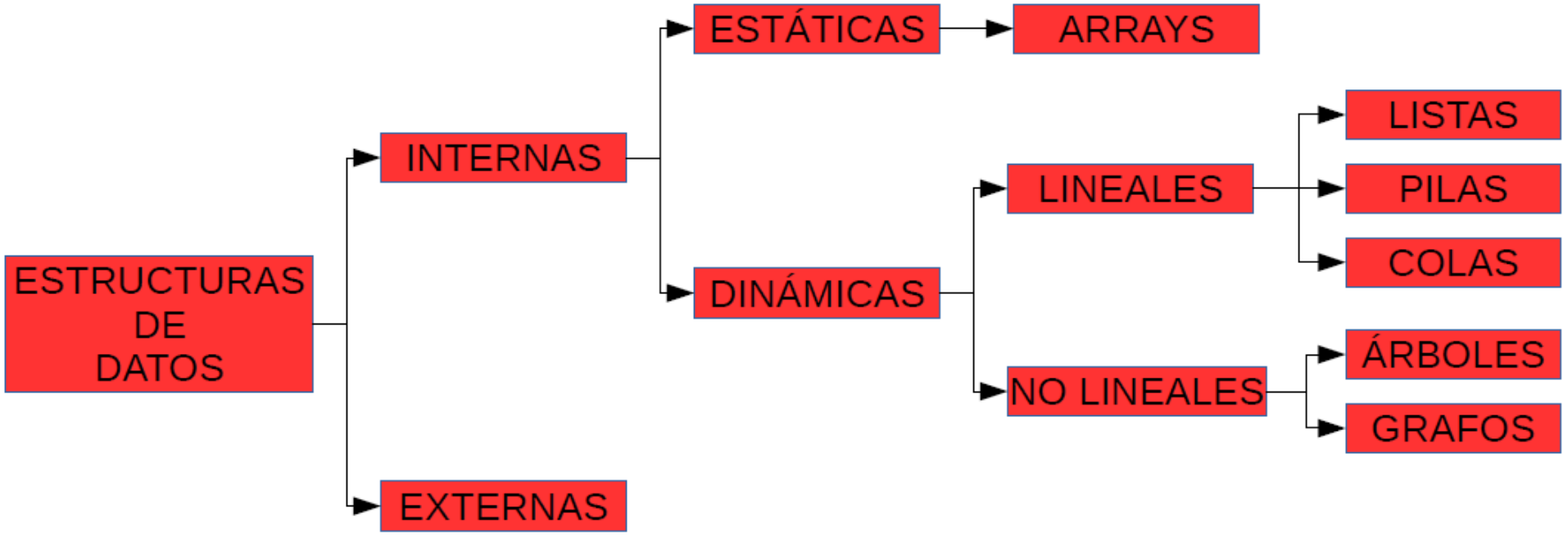
UD5: Almacenamiento de la información en estructuras de datos.

IES LOS SAUCES – BENAVENTE
CFGS DESARROLLO DE APLICACIONES WEB
PROGRAMACIÓN

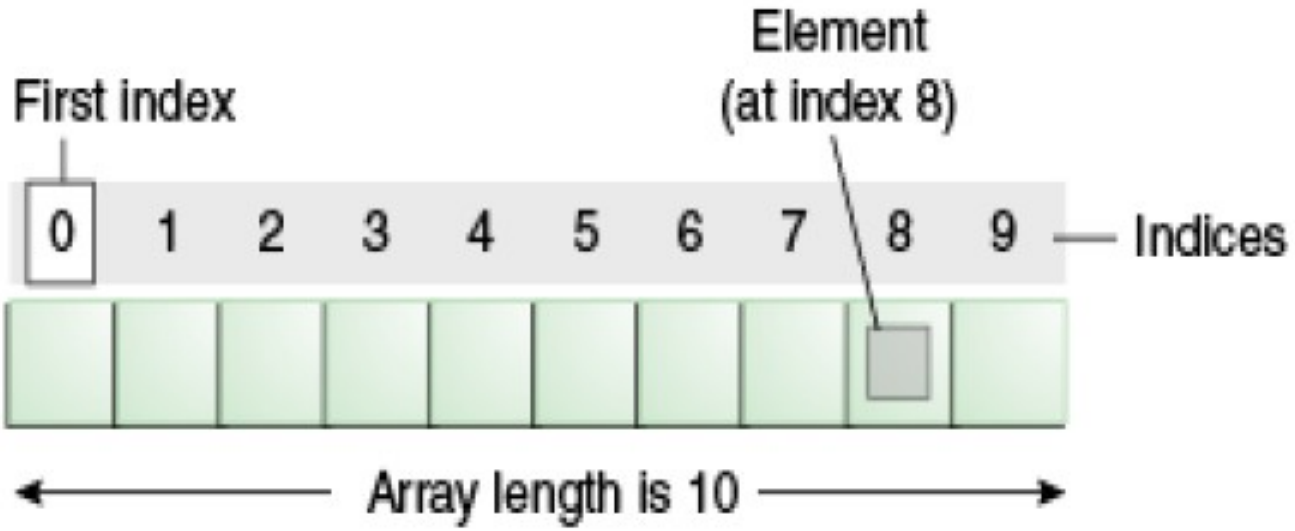
Estructuras de datos

→En programación, una estructura de datos es una forma de organizar un conjunto de datos con el objetivo de facilitar su manipulación.



arrays

→Un array es una estructura de datos formada por una cantidad fija de elementos del mismo tipo, cada uno de los cuales tiene asociado uno, o más índices, que determinan de forma unívoca la posición del elemento en el array.



arrays

Conceptos

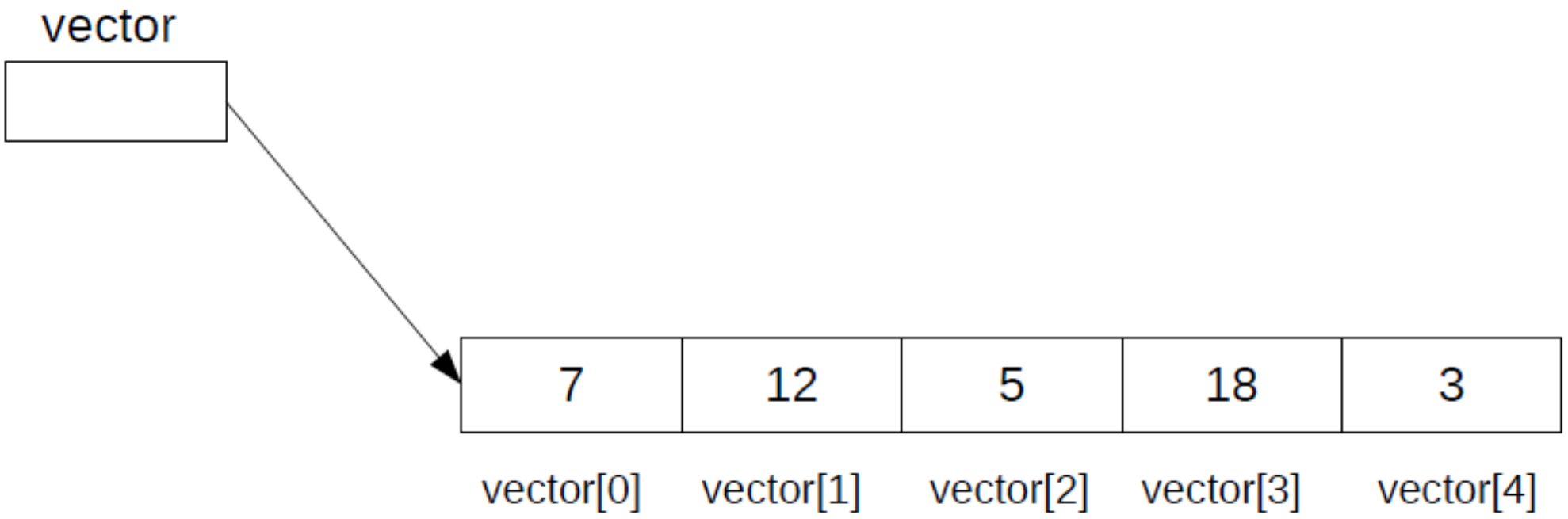
- **Elemento**: cada uno de los datos que forman parte del array.
- **Nombre del array**: identificador utilizado para referenciar el array.
- **Tipo del array**: tipo de dato que es común a cada uno de los elementos. Puede ser un tipo primitivo o referenciado.
- **Tamaño del array**: número máximo de elementos que componen el array.
- **Índice**: valor entero mayor o igual que cero a través del cual podemos acceder directa e individualmente a los distintos elementos de un array.
- **Dimensión de un array**: viene determinada por el número de índices que necesitamos para acceder a cualquiera de sus elementos.

arrays

Tipos de arrays

- **Arrays unidimensionales: para acceder a un elemento es necesario utilizar un índice. También reciben el nombre de vectores.**
- **Arrays bidimensionales: para acceder a un elemento es necesario utilizar dos índices (fila y columna). También reciben el nombre de matrices.**
- **Arrays multidimensionales: para acceder a un elemento es necesario utilizar tres o más índices. También reciben el nombre de poliedros.**

vectores



vectores

Declaración de un array unidimensional

```
tipo[] nombreArray;
```

```
int[] vector;
```

```
tipo nombreArray[];
```

```
int vector[];
```

Creación de un array unidimensional

```
nombreArray=new tipo[numeroElementos];
```

```
vector=new int[5];
```

Inicialización de un array unidimensional

```
tipo[] nombreArray={dato_1,dato_2,…,dato_n};
```

```
int[] vector={7,12,5,18,3};
```

vectores

Tamaño de un array unidimensional

Java considera cada array como un objeto. Un array dispone del atributo length.

```
nombreArray.length
```

```
vector.length
```

Acceder a un elemento de un array unidimensional

```
nombreArray[índice]
```

```
vector[3]
```

Los elementos del array se identifican por la posición que ocupan que van desde la posición cero, que sería el primer elemento, hasta el tamaño del array menos uno.

vectores

- **Recorrido de un array unidimensional**

```
for(int i=0;i<nombreArray.length;i++){
    System.out.println(nombreArray[i]);
}
```

```
for(tipo elemento: nombreArray){
    System.out.println(elemento);
}
```

vectores

→**Pseudocódigo de un algoritmo que permite llenar un vector con diez números enteros solicitados por teclado y los muestra por pantalla.**

```
Proceso Arrays
    Definir vector Como Entero;
    Dimension vector[10];
    Definir i como entero;

    Escribir "Introduce 10 enteros: ";
    Para i<-0 hasta 9 Hacer
        Leer vector(i);
    FinPara

    Para i<-0 hasta 9 Hacer
        Escribir vector(i);
    FinPara
FinProceso
```

vectores

→Codificación Java de un algoritmo que permite llenar un vector con diez números enteros solicitados por teclado y los muestra por pantalla.

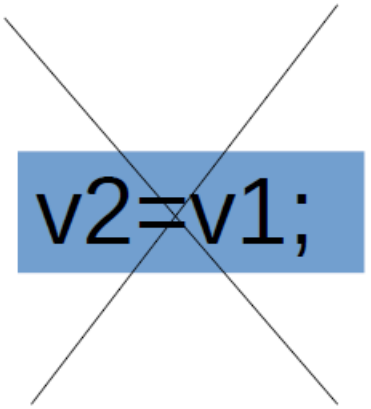
```java
import java.util.Scanner;
public class VectorApp{
    public static void main(String[] args){
        Scanner teclado=new Scanner(System.in);
        int[] vector;

        vector=new int[10];

        System.out.println("Introduce 10 enteros: ");
        for(int i=0;i<vector.length;i++){
            vector[i]=teclado.nextInt();
        }

        System.out.println("Números introducidos: ");
        for(int elemento: vector){
            System.out.println(elemento);
        }
    }
}
```

vectores

- **Copia de un array en otro array**

```
final int LONGITUD=5;
int[] v1={1,2,3,4,5};
int[] v2=new int[LONGITUD];
for(int i=0;i<v1.length;i++)
    v2[i]=v1[i];
}
```

No se puede copiar un array mediante asignación.



vectores

- **Copia de un array en otro array**

Otra forma de copiar un array en otro es mediante el método estático arraycopy de la clase System.

```
public static void arraycopy(Object src,int srcPos,
                    Object dest, int destPos, int length)
```

```
final int LONGITUD=5;
int[] v1={1,2,3,4,5};
int[] v2=new int[LONGITUD];

System.arraycopy(v1,0,v2,0,5);
```

Operaciones sobre vectores

- Búsqueda
  - Búsqueda secuencial
  - Búsqueda binaria
- Ordenación (https://visualgo.net/es/sorting)
  - Intercambio directo (método de la burbuja)
  - Inserción directa (método de la baraja)
  - Selección directa
  - Shell
  - Quicksort
  - Timsort
- Mezcla de dos vectores ordenados

Búsqueda secuencial

Se utiliza cuando el vector no está ordenado.

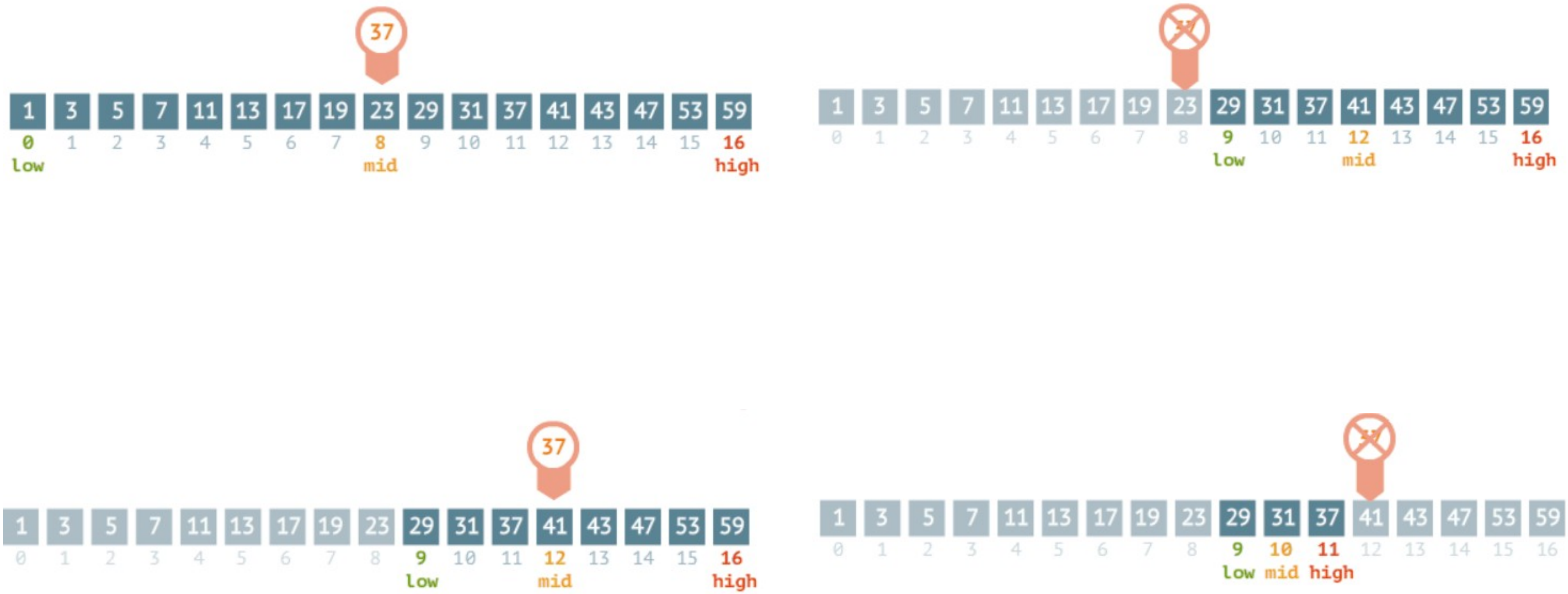
Consiste en buscar un elemento comparándolo secuencialmente con cada elemento del vector hasta encontrarlo, o hasta que se llegue al final.

```
public static int busquedaSecuencial(int[] v, int n){
   int pos=-1;
   for(int i=0;i<v.length && pos==-1;i++){
      if(v[i]==n){
         pos=i;
      }
   }
   return pos;
}
```

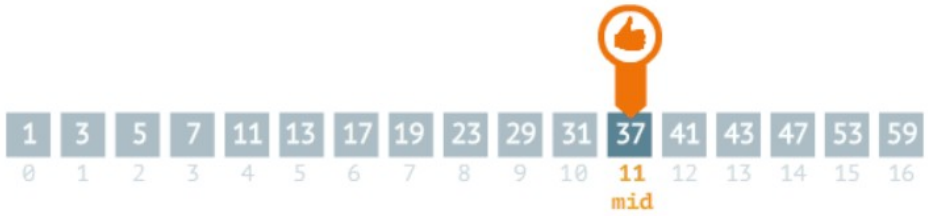
Búsqueda binaria

Se utiliza cuando el vector está ordenado.
Se compara el elemento a buscar con el elemento central del array: si el valor de éste es mayor que el del elemento buscado se repite el procedimiento en la parte del array que va desde el inicio de éste hasta el elemento central, en caso contrario se toma la parte del array que va desde el elemento central hasta el final.

Búsqueda binaria



Búsqueda binaria



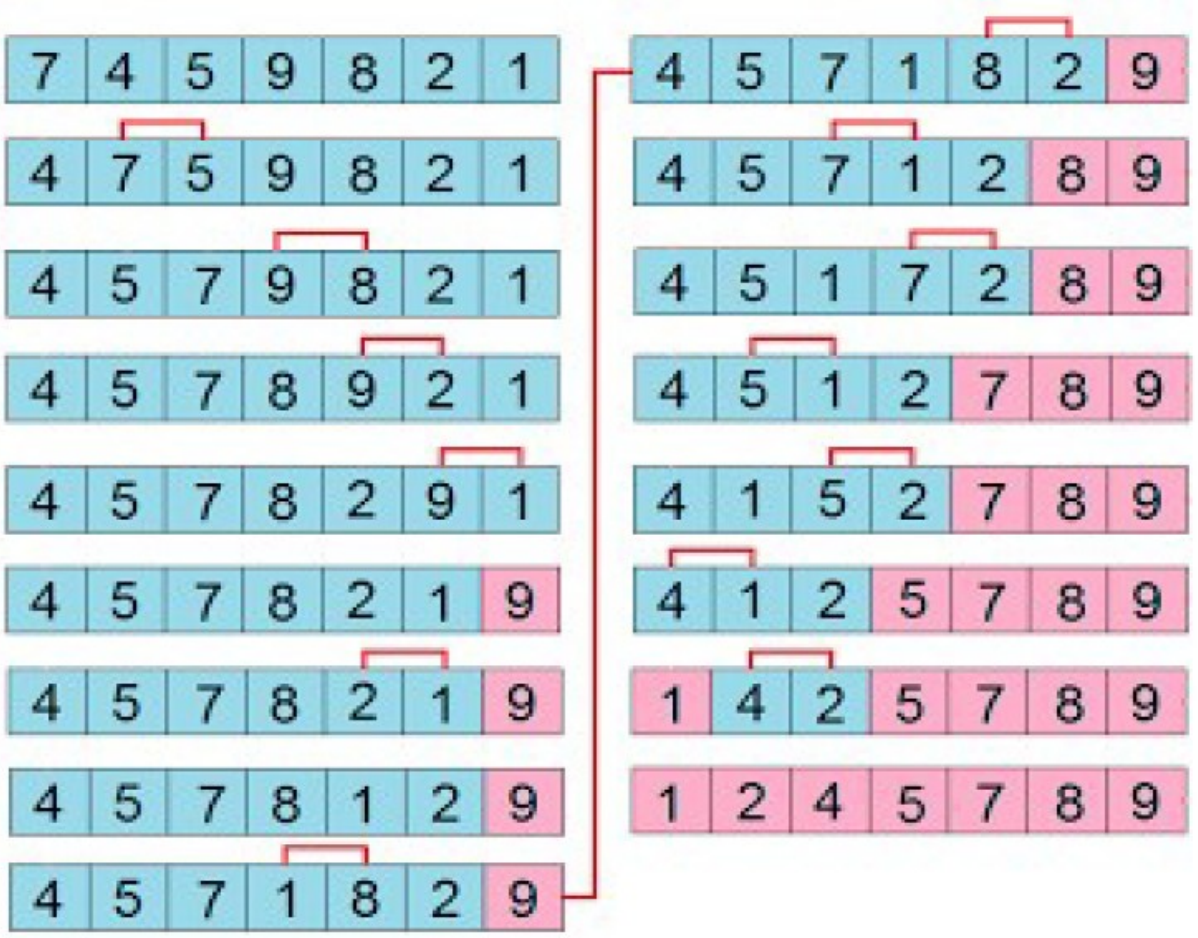
vectores

```
public static int busquedaBinaria(int[] v, int n){
   int pos=-1, inf=0, sup=v.length - 1, centro;
   while(inf <= sup && pos==-1){
      centro=((sup - inf) / 2) + inf;
      if(v[centro] == n){
         pos=centro;
      }
      else{
         if(v[centro] > n){
            sup=centro - 1;
         }
         else{
            inf=centro + 1;
         }
      }
   }
   return pos;
}
```

Intercambio directo

Es un sencillo algoritmo de ordenamiento. También es conocido como el método de la burbuja.
Comenzando desde el inicio del vector, se compara cada par de elementos adyacentes. Si ambos no están ordenados (el segundo es menor que el primero), se intercambian sus posiciones. En cada iteración, un elemento menos necesita ser evaluado (el último), ya que no hay más elementos a su derecha que necesiten ser comparados, puesto que ya están ordenados

Intercambio directo



vectores

```
public static void burbuja(int[] v) {
   boolean cambio = true;
   int aux;
   for(int i = v.length - 1; i > 0 && cambio; i--) {
      cambio = false;
      for (int j = 0; j < i; j++) {
         if (v[j] > v[j+1]) {
            aux = v[j];
            v[j] = v[j+1];
            v[j+1] = aux;
            cambio = true;
         }
      }
   }
}
```

Inserción directa

También es conocido como el método de la baraja.

Consiste en comparar un elemento determinado del vector con todos aquellos de su izquierda e insertarlo si su valor es menor que alguno de ellos.

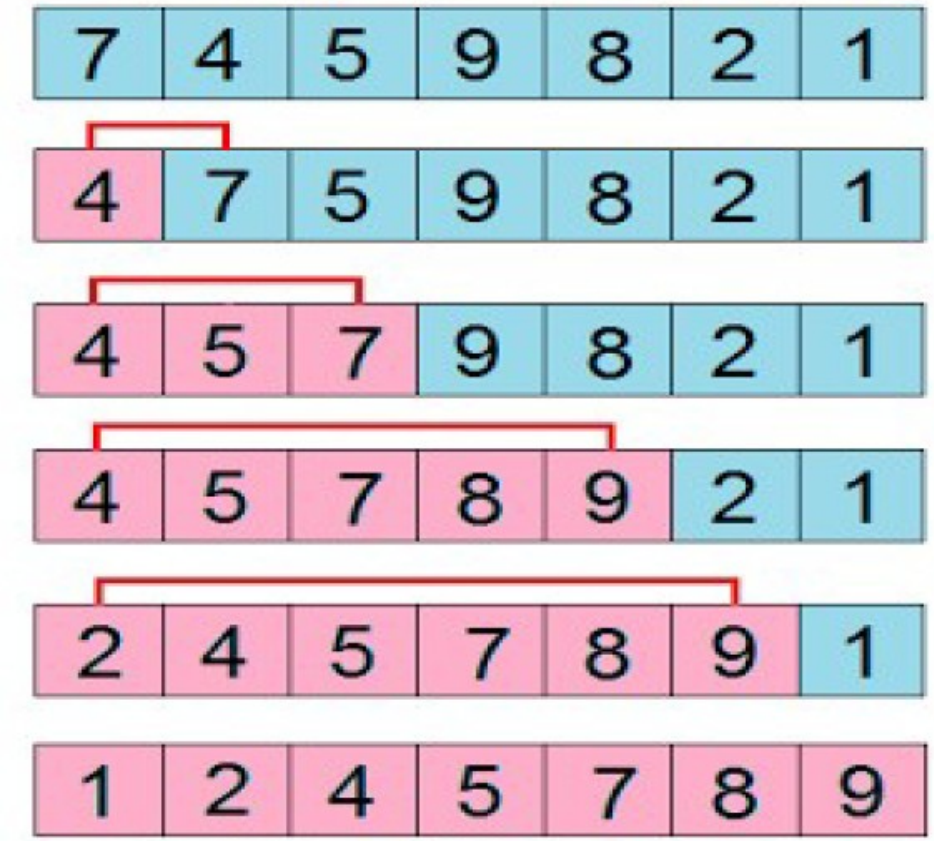
El procedimiento empieza por el segundo elemento del vector y se repite hasta el último

Inserción directa

Por cada elemento se ejecutan los siguientes pasos:

- **Se toma el elemento que estemos considerando y se compara con todos los elementos a su izquierda hasta encontrar uno que sea mayor.**
- **En el lugar ocupado por dicho elemento se inserta el elemento con el que iniciamos el proceso.**
- **Dicha inserción implica desplazar hacia la derecha todos los elementos mayores que el elemento.**

Inserción directa



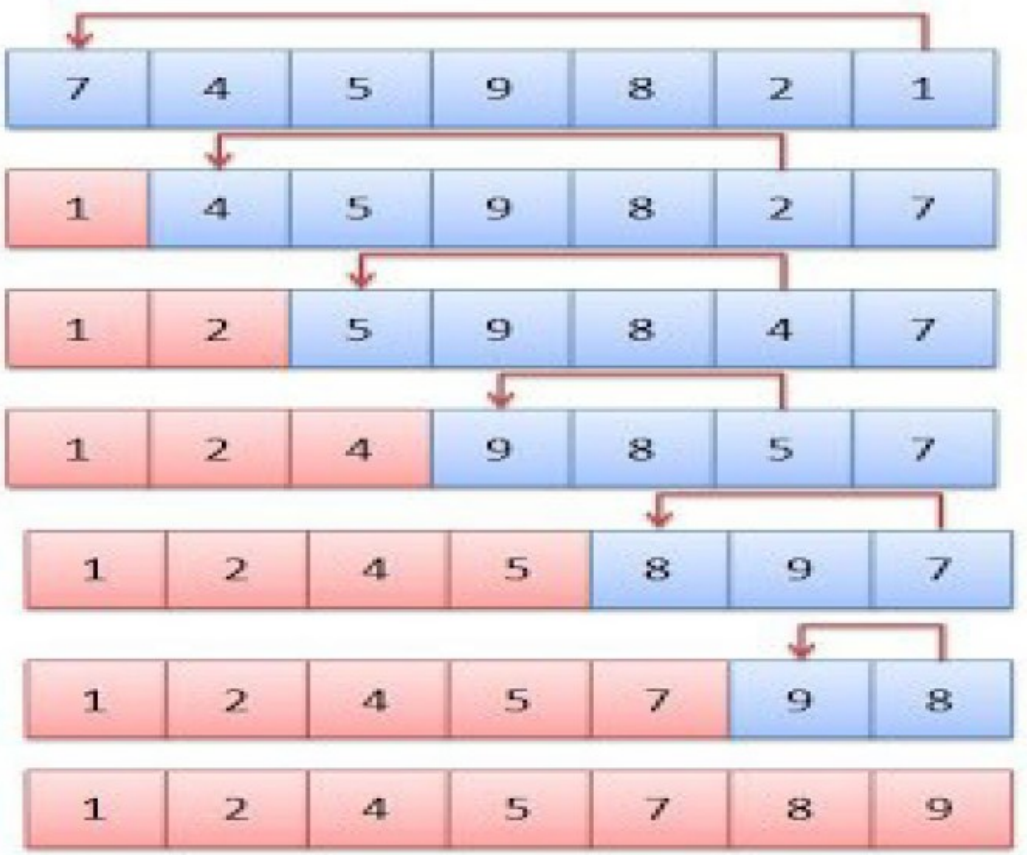
vectores

```
public static void insercionDirecta(int[] v){
    int aux;
    for (int i = 1; i < v.length; i++) {
        aux = v[i];
        for (int j=i; j > 0 && aux < v[j-1];j--){
            v[j] = v[j-1];
        }
        v[j] = aux;
    }
}
```

Selección directa

Consiste en seleccionar el elemento del vector que tenga un valor menor e intercambiarlo por el primer elemento. A continuación se selecciona el menor de los restantes elementos del vector y se intercambia por el segundo. Se trata de repetir esta operación, comparando cada vez un elemento menos, hasta dejar el vector ordenado.

Selección directa



vectores

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

Método Shell

Se considera que el método Shell es una mejora del método de inserción directa.
En el algoritmo de inserción, cada elemento se compara con los elementos contiguos de su izquierda.
El algoritmo Shell modifica los saltos contiguos resultantes de las comparaciones por saltos de mayor tamaño, y con ello se consigue que la ordenación sea más rápida. Generalmente, se toma como salto inicial n/2 (siendo n el número de elementos), y luego se reduce el salto a la mitad en cada repetición hasta que sea de tamaño 1.

vectores

Quicksort

Es un algoritmo basado en la técnica de divide y vencerás.
El algoritmo trabaja de la siguiente forma:
Elegir un elemento del conjunto de elementos a ordenar, al que llamaremos pivote.
Resituar los demás elementos de la lista a cada lado del pivote, de manera que a un lado queden todos los menores que él, y al otro los mayores. Los elementos iguales al pivote pueden ser colocados tanto a su derecha como a su izquierda, dependiendo de la implementación deseada. En este momento, el pivote ocupa exactamente el lugar que le corresponderá en la lista ordenada.
La lista queda separada en dos sublistas, una formada por los elementos a la izquierda del pivote, y otra por los elementos a su derecha.
Repetir este proceso de forma recursiva para cada sublista mientras éstas contengan más de un elemento. Una vez terminado este proceso todos los elementos estarán ordenados.

Timsort

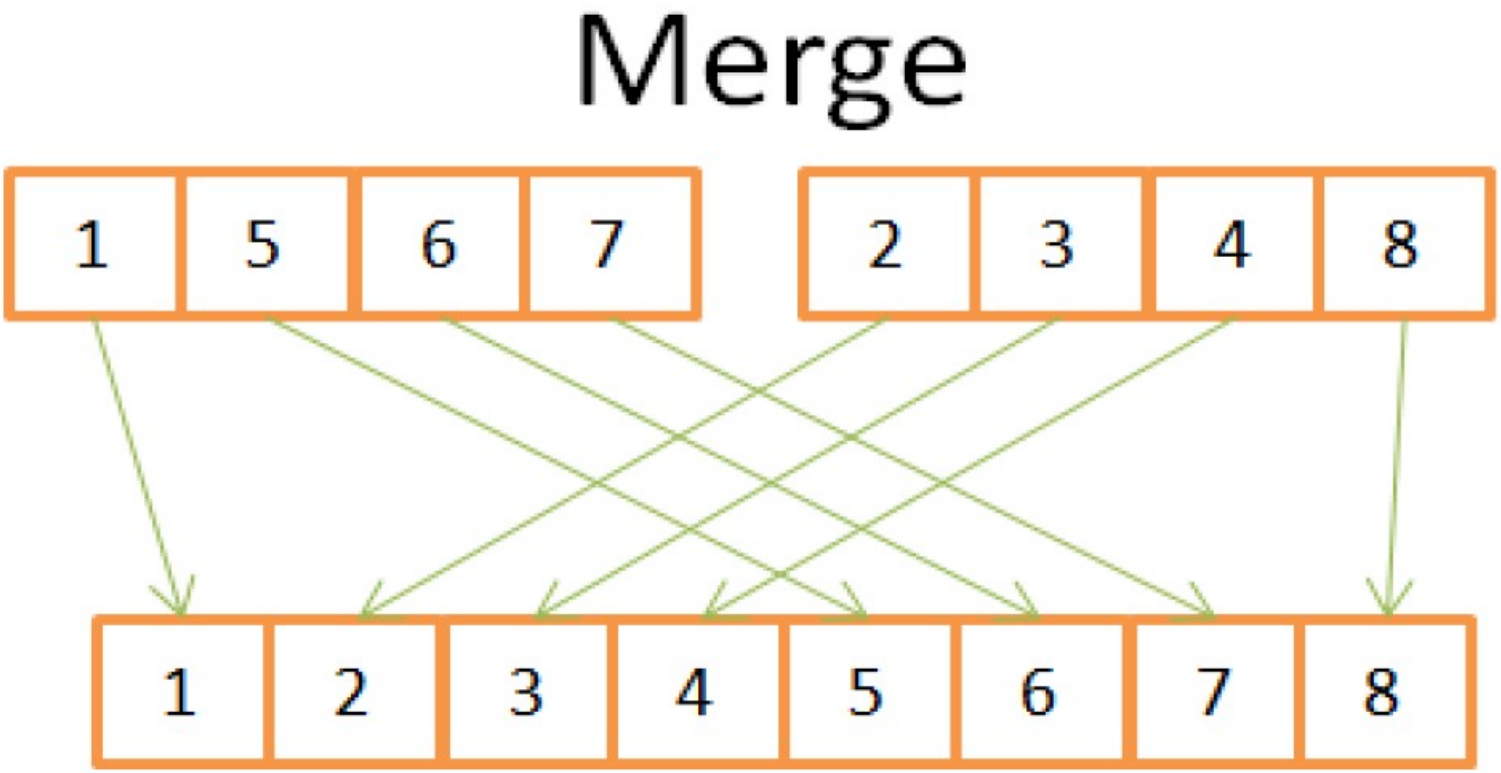
Timsort es un algoritmo de ordenación híbrido estable , derivado de la ordenación por mezcla y la ordenación por inserción, diseñado para funcionar bien en muchos tipos de datos del mundo real.

El algoritmo encuentra subsecuencias de los datos que ya están ordenados y los usa para ordenar el resto de manera más eficiente.

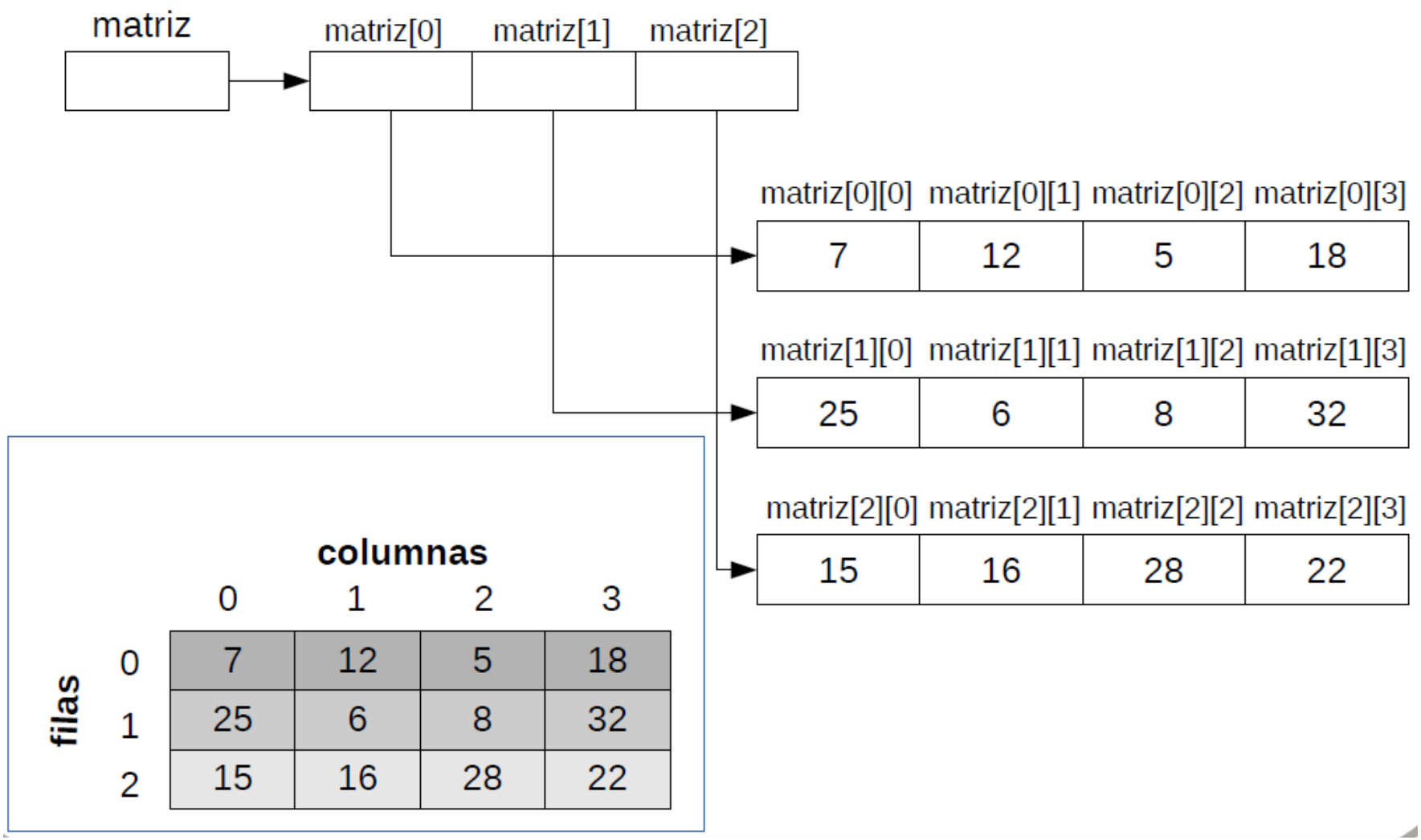
Mezcla de dos vectores

La mezcla consiste en la creación de un array ordenado a partir de otros dos arrays que estén a su vez ordenados según el mismo criterio.
Consiste en ir recorriendo los dos arrays iniciales, comparando un elemento de uno con un elemento de otro. El menor de ambos será copiado en el array final, produciéndose un avance en el vector origen y el destino de la copia. Cuando uno de los dos arrays acaba, se continuarán transfiriendo los elementos restantes del otro array hasta concluirlo también.

Mezcla de dos vectores



matrices


matriz
matriz[0]
matriz[1]
matriz[2]
matriz[0][0] matriz[0][1] matriz[0][2] matriz[0][3]
7 12 5 18
matriz[1][0] matriz[1][1] matriz[1][2] matriz[1][3]
25 6 8 32
matriz[2][0] matriz[2][1] matriz[2][2] matriz[2][3]
15 16 28 22
columnas
0 1 2 3
filas
0 7 12 5 18
1 25 6 8 32
2 15 16 28 22

matrices

Declaración de un array bidimensional

```
tipo[][] nombreArray;
```

```
int[][] vector;
```

```
tipo nombreArray[][];
```

```
int vector[][];
```

Creación de un array bidimensional

```
nombreArray=new tipo[numeroFilas][numeroColumnas];
```

```
vector=new int[3][4];
```

Inicialización de un array bidimensional

```
tipo[][] nombreArray={{11,12,…,1m},
                      {21,22,…,2m},
                      ……….
                      {n1,n2,…,nm}};
```

```
int[][] matriz={{7,12,5,18},
                {25,6,8,32},
                {15,16,28,22}};
```

matrices

- **Creación de un array bidimensional de distinto número de elementos por fila**

```
tipo[][] nombreArray=new tipo[numeroFilas][];
nombreArray[0]=new tipo[numeroElementos1];
nombreArray[1]=new tipo[numeroElementos2];
```

- **Inicialización de un array bidimensional de distinto número de elementos por fila**

```
tipo[][] nombreArray=new tipo[numeroFilas][];
nombreArray[0]={11,12,…,1m};
nombreArray[1]={21,22,…,2n};
```

matrices

Tamaño de un array bidimensional

Número de filas

```
nombreArray.length
```

```
matriz.length
```

Número de columnas

```
nombreArray[índice].length
```

```
matriz[1].length
```

Acceder a un elemento de un array bidimensional

```
nombreArray[fila][columna]
```

```
matriz[1][2]
```

matrices

- **Recorrido de un array bidimensional**

```
for(int i=0;i<nombreArray.length;i++){
    for(j=0;j<nombreArray[i].length;j++){
        System.out.println(nombreArray[i][j]);
    }
}
```

```
for(tipo[] fila: nombreArray){
    for(tipo elemento: fila){
        System.out.println(elemento);
    }
}
```

matrices

- **Pseudocódigo de un algoritmo que permite llenar una matriz de 3 filas y 2 columnas con enteros solicitados por teclado y los muestra por pantalla.**

```
Proceso Matrices
    Definir matriz Como Entero;
    Dimension matriz[3,2];
    Definir i,j como entero;

    Para i<-0 hasta 2 Hacer
        Para j<-0 hasta 1 Hacer
            Escribir "Introduce entero: ";
            Leer matriz(i,j);
        FinPara
    FinPara

    Para i<-0 hasta 2 Hacer
        Para j<-0 hasta 1 Hacer
            Escribir Sin Saltar matriz(i,j);
        FinPara
        Escribir "";
    FinPara
FinProceso
```

matrices

- **Codificación Java de un algoritmo que permite llenar una matriz de 3 filas y 2 columnas con enteros solicitados por teclado y los muestra por pantalla.**

```
import java.util.Scanner;
public class MatrizApp{
    public static void main(String[] args){
        Scanner teclado=new Scanner(System.in);
        int[][] matriz;

        matriz=new int[3][2];

        for(int i=0;i<matriz.length;i++){
            for(int j=0;j<matriz[i].length;j++){
                System.out.print("Introduce entero: ");
                matriz[i][j]=teclado.nextInt();
            }
        }

        System.out.println("Números introducidos: ");
        for(int[] fila: matriz){
            for(int elemento: fila){
                System.out.print(elemento+" ");
            }
            System.out.println();
        }
    }
}
```

array como argumento

- **Paso de un array como argumento de un método**

```
public class PruebaArrays{
   public static void mostrarArray(int[] v){
      for(int i=0;i<v.length;i++){
         System.out.print(v[i] + " ");
      }
   }

   public static void main(String[] args){
      int[] vector={1,2,3,4,5};
      mostrarArray(vector);
   }
}
```

devolviendo un array

- **Devolviendo un array desde un método**

```
public class PruebaArrays{
    public static int[] crearArray(int longitud, int valorIni){
        int[] v=new int[longitud];
        for(int i=0;i<v.length;i++){
            v[i]=valorIni;
        }
        return v;
    }

    public static void main(String[] args){
        int[] vector=crearArray(10,5);
    }
}
```

argumentos main

- **Pasando argumentos al método main**

```
public static void main(String[] args){
   for(String argumento: args){
      System.out.println(argumento);
   }
}
```

Desde línea de comandos:

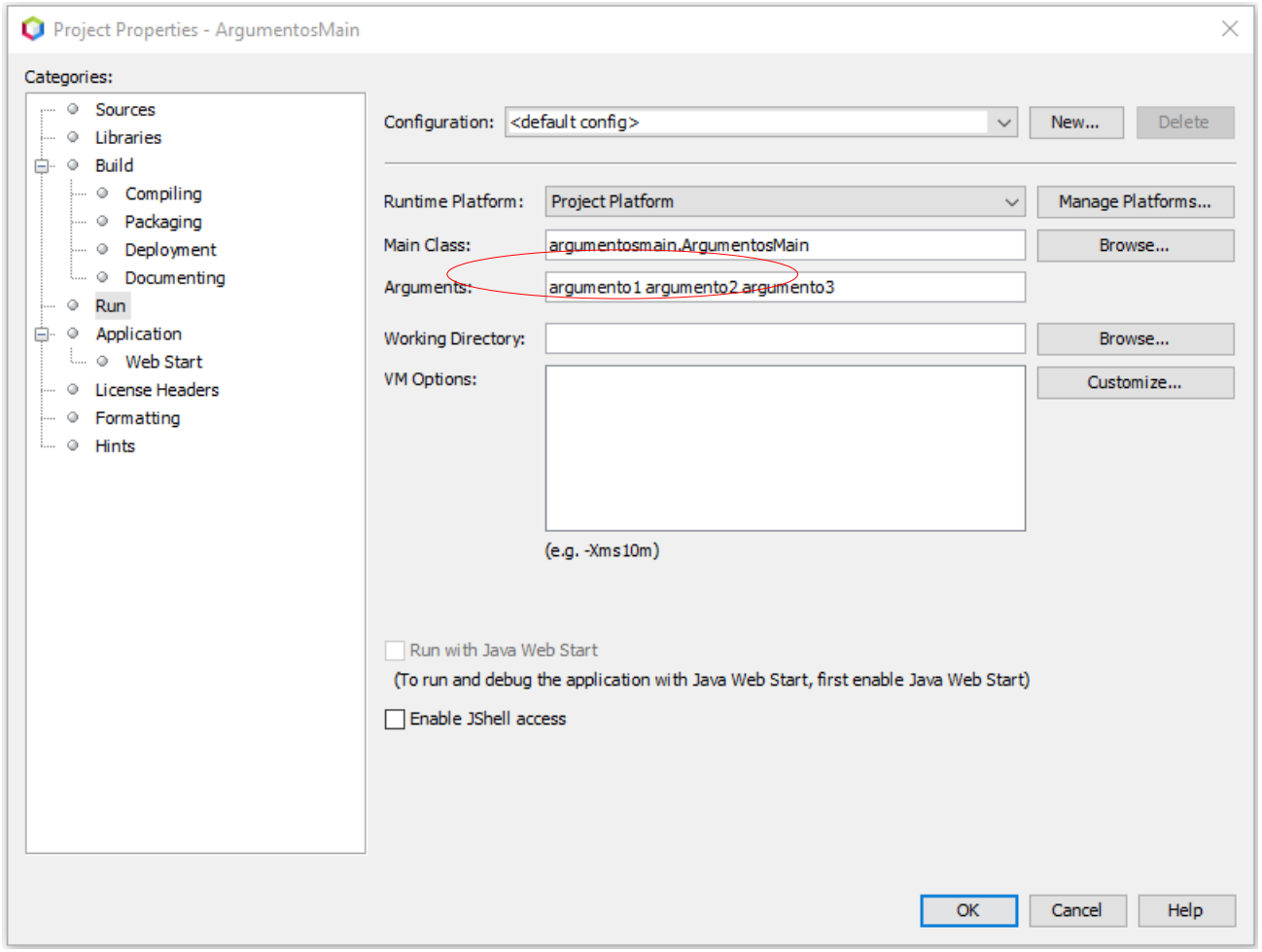
```
java NombreClase argumento1 argumento2 argumento3
```

Salida:

```
argumento1
argumento2
argumento3
```

argumentos main

Desde NetBeans



argumentos variables

Listas de argumentos de longitud variable

Con las listas de argumentos de longitud variable podemos crear métodos que reciben un número arbitrario de argumentos.

Para indicar que el método recibe un número variable de argumentos: `tipo... nombreParam`

En la lista de parámetros del método sólo podrá aparecer una vez y deberá aparecer en último lugar .

argumentos variables

```
public class ArgumentosVariables{

    public static void mostrar(int... param){
        for(int i=0;i<param.length;i++){
            System.out.println(param[i]);
        }
    }

    public static void main(String[] args){
        mostrar(10,20,30,40,50);
    }
}
```

argumentos variables

```
public class ArgumentosVariables{

    public static void mostrar(int... param){
        for(int n: param){
            System.out.println(n);
        }
    }

    public static void main(String[] args){
        mostrar(10,20,30,40,50);
    }
}
```

Arrays

La clase Arrays, perteneciente al paquete java.util, contiene métodos estáticos para la manipulación de arrays.

Los métodos implementan algoritmos de búsqueda, ordenación y de asignación de un valor al array completo.

No se pueden crear objetos de la clase Arrays.

Ordenación de arrays

El método de ordenación, sort(), está sobrecargado de tal forma que se puede ordenar un array de cualquier tipo primitivo.

Para ordenar un array de objetos los elementos deben implementar la interface Comparable.

La ordenación es ascendente.

Implementa Quicksort para los tipos primitivos y TimSort para los objetos.

Ordenación de arrays

`public static void sort(int[] a)`
Ordena el array especificado en orden ascendente.

`public static void sort(int[] a, int fromIndex, int toIndex)`
Ordena el array especificado en orden ascendente desde el elemento especificado por el índice fromIndex (incluido) hasta el elemento especificado por el índice toIndex (excluido).

Búsqueda en arrays

El método de búsqueda, binarySearch(), está sobrecargado de tal forma que se puede buscar en un array de cualquier tipo primitivo.

Para buscar en un array de objetos los elementos deben implementar la interface Comparable.

La búsqueda se realiza en un array ordenado.

Implementa el algoritmo de búsqueda binaria.

Búsqueda en arrays

`public static int binarySearch(int[] a, int key)`
Busca en el array especificado el valor especificado usando el algoritmo de búsqueda binaria. El array debe estar ordenado. Devuelve la posición en la que se encuentra el elemento buscado, si está. Si no está devuelve un número negativo.

`public static int binarySearch(int[] a, int fromIndex, int toIndex, int key)`
Busca en el array especificado el valor especificado desde el elemento especificado por el índice fromIndex (incluido) hasta el elemento especificado por el índice toIndex (excluido).

Asignación de un elemento en arrays

El método de asignación, fill(), está sobrecargado de tal forma que se puede asignar un elemento a todas las posiciones de un array de cualquier tipo primitivo y Object.

También se puede asignar un elemento a un rango de posiciones del array.

Asignación de un elemento en arrays

`public static void fill(int[] a, int val)`
Asigna el valor especificado a cada elemento del array.

`public static void fill(int[] a, int fromIndex, int toIndex, int val)`
Asigna el valor especificado a cada elemento del rango del array. El rango va desde el elemento especificado por el índice fromIndex (incluido) hasta el elemento especificado por el índice toIndex (excluido).

Copia de arrays (métodos sobrecargados)

`public static int[] copyOf(int[] original, int newLength)`

Copia el array especificado, truncando o rellenando con ceros (si es necesario) para que la copia tenga la longitud especificada.

`public static int[] copyOfRange(int[] original, int from, int to)`

Copia el rango especificado del array especificado.

Comparación de arrays (método sobrecargado)

`public static boolean equals(int[] a1, int[] a2)`
Devuelve true si los dos arrays son iguales. Dos arrays son iguales si contienen los mismos elementos en el mismo orden.

Comparación de arrays

`public static boolean deepEquals(Object[] a1,Object[] a2)`
Devuelve true si los dos arrays son profundamente iguales entre sí. Dos arrays se consideran profundamente iguales si ambos son nulos o si contienen el mismo número de elementos y todos los elementos son profundamente iguales.
Nos permite comprobar la igualdad en arrays multidimensionales.

Arrays

Representación de arrays en forma de cadena(método sobrecargado)

`public static String toString(int[] a)`
Devuelve una cadena que representa el array especificado.

Representación de arrays en forma de cadena

`public static String deepToString(Object[] a)`
Devuelve una cadena que representa el contenido profundo del array especificado.
Este método está diseñado para convertir arrays multidimensionales en cadenas.