

UNIDAD 3.

Estructuras de datos.



Índice

1. Ficha unidad didáctica.....	1
2. Contenidos.....	2
2.1. Introducción.....	2
2.2. Enumerados.....	2
2.3. Vectores	6
2.3.1 Algoritmos de búsqueda.....	7
2.3.2 Algoritmos de ordenación.....	9
2.3.3 Burbuja.....	11
2.3.4 Selección.....	12
2.3.5 Inserción.....	14
2.4. Cadenas de caracteres.....	15
2.5. Arrays multidimensionales.....	18
2.5.1 Paso de parámetros a programas.....	22
2.5.2 Conversiones.....	24
2.6. Estructuras.....	24
2.7. Estructuras y vectores.....	28
3. Actividades y ejercicios.....	29

1. Ficha unidad didáctica.

OBJETIVOS DIDÁCTICOS	
<p>OD1: Exponer la necesidad de estructuras de datos.</p> <p>OD2: Enunciar las características de las estructuras básicas.</p> <p>OD3: Usar de forma razonada las diferentes estructuras de datos básicas.</p> <p>OD4: Utilizar correctamente las estructuras básicas proporcionadas por el lenguaje .</p> <p>OD5: Evaluar la importancia de planificar temporalmente el trabajo a desarrollar.</p> <p>OD6: Selecciona correctamente las estructura en función del problema a a solucionar.</p> <p>EV3. Advertir de la importancia del trabajo en grupo en el ámbito empresarial.</p> <p>RL2. Sensibilizar sobre las consecuencias de una vida y trabajo sedentario para la salud.</p> <p>TIC1. Emplear Internet para obtener información técnica necesaria para la realización de la actividad laboral.</p> <p>TIC2. Favorecer el uso de Internet como forma de actualización en nuevas tecnologías relacionadas con el ámbito laboral de la informática.</p>	
RESULTADOS DE APRENDIZAJE	RA6
CONTENIDOS	
<p>Enumerados.</p> <p>Vectores.</p> <p>Cadenas de caracteres.</p> <p>Arrays multidimensionales.</p> <p>Estructuras.</p> <p>Estructuras y vectores.</p>	
ORIENTACIONES METODOLÓGICAS	
<p>La planificación de las actividades se realiza teniendo en cuenta la problemática de comprensión del concepto abstracto de vectores y matrices.</p> <p>Las actividades de introducción e investigación tienen un gran componente para desarrollar la capacidad de “aprender a aprender”.</p> <p>Las actividades se dirigen a potenciar la documentación de trabajos realizados y lectura de documentación técnica.</p>	
CRITERIO DE EVALUACIÓN	6a

2. Contenidos.

2.1. Introducción.

En estos momentos del curso se intuye que simplemente usando tipos de datos primitivos y estructuras de control para el manejo de grandes cantidades de información homogénea es prácticamente imposible, además muchas de las expresiones de las estructuras de control se toman de un valor entero que tiene un significado intrínseco. Se hace necesario la existencia de otras estructuras que amplíen a los tipos básicos o primitivos y resuelvan el manejo de grandes tamaños de datos homogéneos y la identificación en las opciones de forma clara y no numérica. En la unidad se tratan los enumerados, el concepto de vector, introduciendo las clases a través de la clase "String" que se puede ver como un vector de tipo carácter, los vectores multidimensionales, las estructuras de datos y finalizando con la integración de los vectores y las estructuras.



Java no posee estructuras como tal, en comparación con otros lenguajes como C, C++ o incluso PHP, para simular las estructuras se utilizarán clases publicas.

```
struct mystruct
{
    char  a;
    int   b;
    float c;
};

struct mystruct myvar;

myvar.b = 99;
```

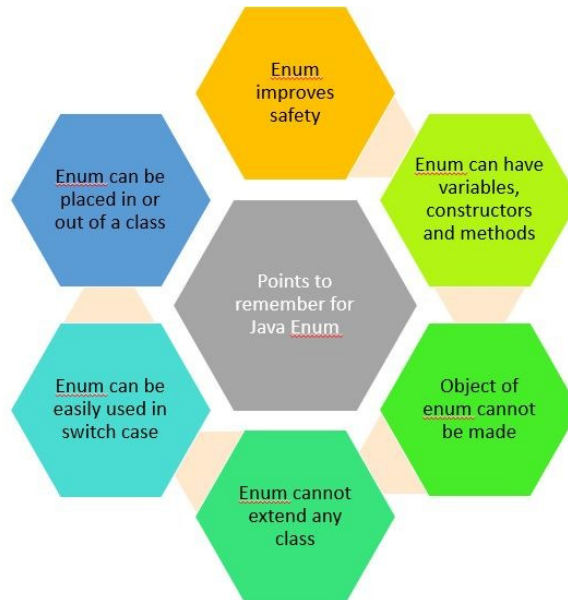
```
struct Employee {
    string $firstName;
    string $lastName;
    int $salary;
    bool $fullTime;
}
```

Ejemplo de código en C y PHP respectivamente.

2.2. Enumerados.

Los enumerados permiten la **personalización** de tipos de datos primitivos, de forma que se definan un conjunto posible de valores que se asocia a otro conjunto de cadenas o nombres de uno en uno. Esto facilita el mantenimiento y el modelado de las aplicaciones.

Un tipo enumerado en Java es un tipo “especial” que en cierta medida puede usarse como una clase y admite ciertas posibilidades especiales. Debido a que los tipos enumerados tiene características de clase especial en Java, hay muchas cosas que se pueden realizar dentro de un enum, además de declarar constantes, un tipo enumerado puede contener métodos y variables.



Un ejemplo sencillo de enumerados:


```
public enum Estado {  
  
    /**  
     * Indica que el proceso no ha iniciado.  
     */  
    POR_INICIAR,  
    /**  
     * Indica que el proceso inicio pero no ha finalizado.  
     */  
    EN_PROGRESO,  
    /**  
     * Indica que el proceso finalizo correctamente.  
     */  
    FINALIZADO,  
    /**  
     * Indica que el proceso finalizo con errores.  
     */  
    ERROR;  
}
```

Ahora es posible utilizar estos enumerados en el código, algunos ejemplos de uso:

```
public static void main(String args[]) {  
    //variable de tipo Estado  
    Estado estado;  
    //para almacenar la entrada de teclado  
    String opcion = null;  
    System.out.println("Introduce estado");  
    //en este caso ya no es leer un tipo primitivo, sino un objeto cadena  
    BufferedReader lector = new BufferedReader(  
        new InputStreamReader(System.in));  
    /*es una operación que da fallo, Java obliga a tratar los posibles  
fallos con un excepción */  
    try {  
        opcion = lector.readLine();  
    } catch (IOException ex) {  
        Logger.getLogger(Ejemplo.class.getName()).log(Level.SEVERE, null,  
ex);  
    }  
    //se transforma la cadena  
    estado = Estado.valueOf(opcion);  
    //se puede tener tanto el nombre, como el numero  
    System.out.println("El nombre que lo identifica es " + estado.name());  
    System.out.println("El ordinal que lo identifica es " +  
estado.ordinal());  
    //se puede hacer referencia en las estructuras de control por el nombre  
    switch (estado) {  
        case POR_INICIAR:  
            break;  
        case EN_PROGRESO:  
            break;  
        case FINALIZADO:  
            break;  
        case ERROR:  
            break;  
        default:  
            throw new AssertionError(estado.name());  
    }  
    //un tipo especial de for para objetos, se trata en temas posteriores  
    for (Estado p : Estado.values()) {  
        System.out.printf("Los valores son" + p.name() + " y los ordinales  
" + p.ordinal());  
    }  
}
```

Como se ha indicado los tipos enumerados son clases y es posible ampliar la funcionalidad de los enumerados, un ejemplo más complejo:

```
public enum Equipo
{
    BARÇA("FC Barcelona",1) ,
    REAL_MADRID("Real Madrid",2) ,
    SEVILLA("Sevilla FC",4) ,
    VILLAREAL("Villareal",7);
    private String nombreClub;
    private int puestoLiga;
    //constructor tema 4, da los valores iniciales
    private Equipo (String nombreClub, int puestoLiga){
        this.nombreClub = nombreClub;
        this.puestoLiga = puestoLiga;
    }
    public String getNombreClub() {
        return nombreClub;
    }
    public int getPuestoLiga() {
        return puestoLiga;
    }
}
```

 Los enumerados en Java tienen muchas características de clases, en este punto del curso aun no se conoce dichas características, lo importante es que es posible asociar a un nombre una variable de forma que se le otorgue significado y se pueda utilizar en condiciones además de limitar los posibles valores que toma.



[Enumerados en Java. UPV.](#)



Se desea gestionar la inscripción de una carrera popular, las categorías son: Infantil, Cadete, Juvenil, General y Senio, crear un enumerado para la gestión y usar en un switch para solicitar la categoria, para mostrar usar el for.

2.3. Vectores .

También conocidos como Arrays. Un vector es un medio de guardar un conjunto de objetos del mismo tipo o clase. Se accede a cada elemento individual del array mediante un número entero denominado índice. 0 es el índice del primer elemento y n-1 es el índice del último elemento, siendo n, la dimensión del array. Los arrays son objetos en Java, las operaciones principales con los arrays son:

- Declarar el array
- Crear el array
- Inicializar los elementos del array
- Usar el array

Declarar y crear un array.

Para declarar un array se escribe

```
tipo_de_dato[] nombre_del_array;
```

Para declarar un array de enteros se escribe:

```
int[] numeros;
```

Para crear un array de 4 número enteros (se reserva memoria):

```
numeros=new int[4];
```

La declaración y la creación del array se puede hacer en una misma línea.

```
int[] numeros =new int[4];
```

Inicializar y usar los elementos del array

Es posible inicializarlo al mismo tiempo que se crea:

```
int [] numeros={1,2,3,4};
```

O de forma individual, elemento por elemento:

```
numeros[0]=2;  
numeros[1]=-4;  
numeros[2]=15;  
numeros[3]=-25;
```

Se pueden inicializar en un bucle for como resultado de alguna operación


```
for(int i=0; i<4; i++){  
    numeros[i]=i*i+4;  
}
```

Además, es posible conocer la longitud de un array con el atributo `length`. Un ejemplo de sus uso:

```
for(int i=0; i<numeros.length; i++){  
    numeros[i]=i*i+4;  
}
```

Existe la posibilidad de recorrer los array (y otras estructuras con el conocido como `ForEach`):

```
for (int i: numeros) {  
    System.out.println (i);  
}
```

Los arrays se pueden declarar, crear e inicializar en una misma línea, del siguiente modo

```
int[] numeros={2, -4, 15, -25};
```

Para imprimir a los elementos de array nombres se escribe

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



Ejercicios para practicar.

1. **Crear un programa que solicite 10 números enteros y los almacene en un array. Una vez almacenados obtener:**
 1. El mayor.
 2. El menor.
 3. La suma total.
 4. La media.
2. Crear un array de 50 números cuyos valores se encuentren entre 0 y 9, para iniciar los valores de forma automática usar la siguiente expresión:

```
(int) (Math.random()*10)
```

- Obtener en otro array el número de veces que aparece cada número de 50 elementos.

2.3.1 Algoritmos de búsqueda.

Una de las tareas más usuales en los vectores es la de búsqueda de un elemento. Existen 2 algoritmos o métodos de búsqueda.

- Secuencial.
- Binaria o dicotómica.

La primera consiste en recorrer el array desde el inicio hasta el final o encontrar el elemento buscado, su complejidad es lineal, ya que depende del tamaño del vector a buscar, un ejemplo del algoritmo:

```
int[] vector={4,75,86,87,32,3,78,56,100,2,45,78,1};
int numero,i;
boolean encontrado=false;
Scanner teclado= new Scanner(System.in);
System.out.println("Introduce el número a buscar");
numero=teclado.nextInt();
for(i=0;i<vector.length && !encontrado;i++){
    if(vector[i]==numero){
        encontrado=true;
    }
}
if(encontrado){
    System.out.println("El número "+numero+ "se encuentra en la
posición "+i);
}else{
    System.out.println("El número "+numero+ " NO se encuentra");
}
```

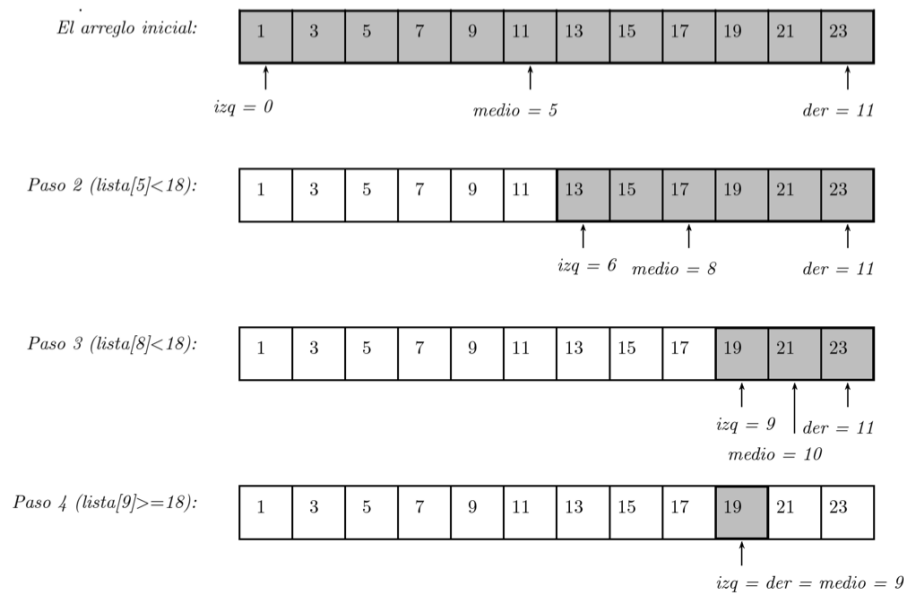
La segunda, binaria o dicotómica ha de cumplir la condición principal de que el vector se encuentre **ordenado**, se basa en ir “partiendo” el vector en 2 en cada iteración en función de si el elemento central es menor o mayor que el elemento buscado:

```
int[] vector = {14, 25, 36, 47, 52, 53, 68, 76, 100, 112, 145, 178, 541};
int numero, inferior, superior, mitad;
boolean encontrado = false;
```

```
Scanner teclado = new Scanner(System.in);
System.out.println("Introduce el número a buscar");
numero = teclado.nextInt();
inferior = 0;
superior = vector.length-1;

    System.out.println("Al iniciar inferior:"+inferior+"
superior:"+superior);
    do {
        mitad = (superior + inferior) / 2;
        if (numero == vector[mitad]) {
            encontrado = true;
        } else {
            if (numero < vector[mitad]) {
                superior = mitad-1 ;
            } else {
                inferior = mitad+1 ;
            }
        }
    }

    System.out.println("Al salir vale mitad:"+mitad+"
inferior:"+inferior+" superior:"+superior);
    } while (inferior <= superior  && !encontrado);
    if (encontrado) {
        System.out.println("El número " + numero + "se encuentra en la
posición      " + mitad);
    } else {
        System.out.println("El número " + numero + " NO se encuentra");
    }
}
```



Algoritmo búsqueda binaria

La complejidad temporal de este algoritmo es logarítmica en concreto log en base 2 de n, mejor que el lineal.



Ejercicios para practicar.

1. Crear un array de 20 elementos enteros aleatorios no repetidos y realizar una búsqueda secuencial solicitando el elemento a buscar por teclado, en caso de no existir el elemento mostrar un mensaje.
2. Definir un array de enteros no repetidos ordenado y a partir del mismo realizar una búsqueda binaria.

Java posee un método en los arrays (se trata en la unidad 4 el concepto de método) que realiza una búsqueda binaria en un array sin necesidad de implementar el algoritmo. Se encuentra en el paquete `java.util.Arrays` y al ejecutarlo devuelve -1 en caso de no encontrarlo y el índice en el caso de encontrarlo.

```
resultado=(java.util.Arrays.binarySearch(vector, 541));
```

Se puede consulta la documentación en:
<https://docs.oracle.com/javase/8/docs/api/java/util/Arrays.html>

Por defecto es capaz de ordenar datos primitivos, pero con modificaciones es posible ordenar cualquier tipo de dato (se trata en temas posteriores).



Ejercicios para practicar.

1. Modificar el ejercicio 2 anterior para realizar una búsqueda binaria usando la técnica anterior.

2.3.2 Algoritmos de ordenación.

Es muy común necesitar ordenar vectores, por ejemplo apellidos por orden alfabético, importe de facturas o preparar array para búsqueda binaria. Existen muchos algoritmos de ordenación, los más sencillos con complejidades de n^2 , y los mejores con complejidades del constantes como el RadixSort.

[Algoritmo Radix Sort.](#)

Array Sorting Algorithms

Algorithm	Time Complexity			Space Complexity
	Best	Average	Worst	Worst
Quicksort	$\Omega(n \log(n))$	$\Theta(n \log(n))$	$O(n^2)$	$O(\log(n))$
Mergesort	$\Omega(n \log(n))$	$\Theta(n \log(n))$	$O(n \log(n))$	$O(n)$
Timsort	$\Omega(n)$	$\Theta(n \log(n))$	$O(n \log(n))$	$O(n)$
Heapsort	$\Omega(n \log(n))$	$\Theta(n \log(n))$	$O(n \log(n))$	$O(1)$
Bubble Sort	$\Omega(n)$	$\Theta(n^2)$	$O(n^2)$	$O(1)$
Insertion Sort	$\Omega(n)$	$\Theta(n^2)$	$O(n^2)$	$O(1)$
Selection Sort	$\Omega(n^2)$	$\Theta(n^2)$	$O(n^2)$	$O(1)$
Tree Sort	$\Omega(n \log(n))$	$\Theta(n \log(n))$	$O(n^2)$	$O(n)$
Shell Sort	$\Omega(n \log(n))$	$\Theta(n(\log(n))^2)$	$O(n(\log(n))^2)$	$O(1)$
Bucket Sort	$\Omega(n+k)$	$\Theta(n+k)$	$O(n^2)$	$O(n)$
Radix Sort	$\Omega(nk)$	$\Theta(nk)$	$O(nk)$	$O(n+k)$
Counting Sort	$\Omega(n+k)$	$\Theta(n+k)$	$O(n+k)$	$O(k)$
Cubesort	$\Omega(n)$	$\Theta(n \log(n))$	$O(n \log(n))$	$O(n)$

Complejidades algoritmos ordenación.

Estos algoritmos se encuentran implementados en la mayor parte de los lenguajes, ya sea de forma nativa o incluyendo librerías externas, por ejemplo Java utiliza por defecto el algoritmo Dual-Pivot Quicksort en el paquete `java.util.Arrays`.

```
int[] vector={14,25,3,7,52,5,68,6,10,12,15,78,1};
java.util.Arrays.sort(vector);
for (int i:vector){
    System.out.print(i+" ");
}
```

```
System.out.println() ;
```

El resultado es el siguiente:

```
1 3 5 6 7 10 12 14 15 25 52 68 78P
```

Pero por desgracia un buen programador ha de conocer, entender, implementar, depurar y modificar los algoritmos de ordenación más sencillos y conocidos con una complejidad n^2 , ya que necesitan realizar 2 pasadas completas por todo el vector para ordenar, siendo estos:

- Burbuja.
- Selección.
- Inserción.

Rara vez se utilizan estos algoritmos por la complejidad temporal que implica utilizándose otros como el Quicksort, aunque pasa su uso es necesario utilizar estructuras de datos más complejas como listas que se explican en temas posteriores

2.3.3 Burbuja.

Consiste en recorrer el array comparando elementos adyacentes, en caso de querer ordenar de menor a mayor, si el elemento i es menor que el elemento $i+1$, se intercambian. Es necesario recorrer el array tantas veces como elementos tenga el array. Se denomina burbuja ya que los elementos mayores “burbujean” hacia el final del vector.

Pasada 1

a[0]	a[1]	a[2]	a[3]	a[4]
20	40	50	30	80

Ordenados 20 y 40

a[0]	a[1]	a[2]	a[3]	a[4]
20	40	50	30	80

Ordenados 40 y 50

a[0]	a[1]	a[2]	a[3]	a[4]
20	40	50	30	80

Intercambio 50 y 30

a[0]	a[1]	a[2]	a[3]	a[4]
20	40	30	50	80

50 y 80 elementos mayores y ordenados

Ejemplo algoritmo burbuja.

El algoritmo es el siguiente:

```
int[] vector = {14, 25, 3, 7, 52, 5, 68, 6, 10, 12, 15, 78, 1};
int temporal;
for (int i = 0; i < vector.length; i++) {
    for (int j = 0; j < vector.length; j++) {
        if (vector[i] < vector[j]) {
            temporal = vector[i];
            vector[i] = vector[j];
            vector[j] = temporal;
        }
    }
}
```

Observar que para no perder ningún valor se define la variable temporal (se puede llamar de cualquier otra forma) para almacenar poder realizar el intercambio sin perder el valor.

El resultado es:

1 3 5 6 7 10 12 14 15 25 52 68 78



¿Qué modificación es necesaria introducir para ordenar de mayor a menor?



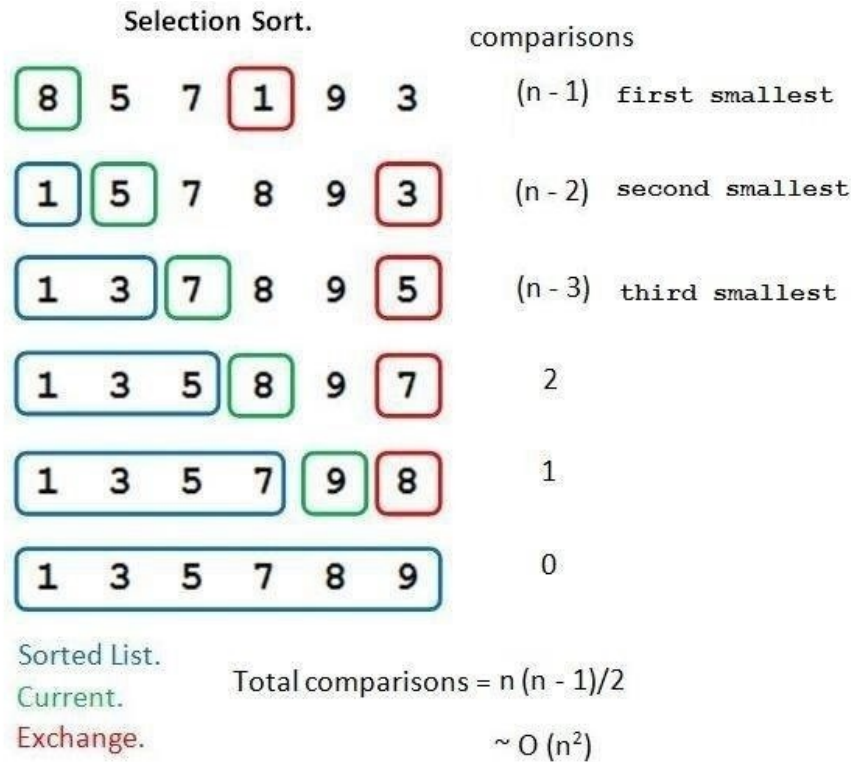
[Algoritmo ordenación burbuja UPV.](#)

2.3.4 Selección.

Se basa en dividir el vector en 2, una parte ordenada y otra desordenada, la primera situada a la izquierda y la segunda a la derecha, se busca el elemento más pequeño en la parte derecha y se intercambia por el primer elemento de la parte derecha sin ordenar, a partir de ese momento la parte ordenada tiene un elemento más ordenado.



¿Qué complejidad tiene el algoritmo? Compararlo con la burbuja.



Ejemplo algoritmo selección.

El algoritmo en Java:

```
int[] vector = {14, 25, 3, 7, 52, 5, 68, 6, 10, 12, 15, 78, 1};
//indice sirve para guardar el menor de la parte desordenada, temporal
//para el intercambio
int indice, temporal;
//indica la parte ordenada
int indice_ordenado = 0;
for (int i = indice_ordenado; i < vector.length; i++) {
    indice = i;
    for (int j = indice_ordenado ; j < vector.length; j++) {
        //si se encuentra uno menor se guarda para intercambiarlo
        if (vector[indice] > vector[j]) {
            indice = j;
        }
    }
    //se realiza el intercambio si se ha encontrado uno ordenado después
    if (indice_ordenado != indice) {
        temporal = vector[indice_ordenado];
        vector[indice_ordenado] = vector[indice];
        vector[indice] = temporal;
    }
}
```

```
//se incrementa la parte ordenada en 1  
indice_ordenado++;  
}
```

El resultado es:

1	3	5	6	7	10	12	14	15	25	52	68	78
---	---	---	---	---	----	----	----	----	----	----	----	----



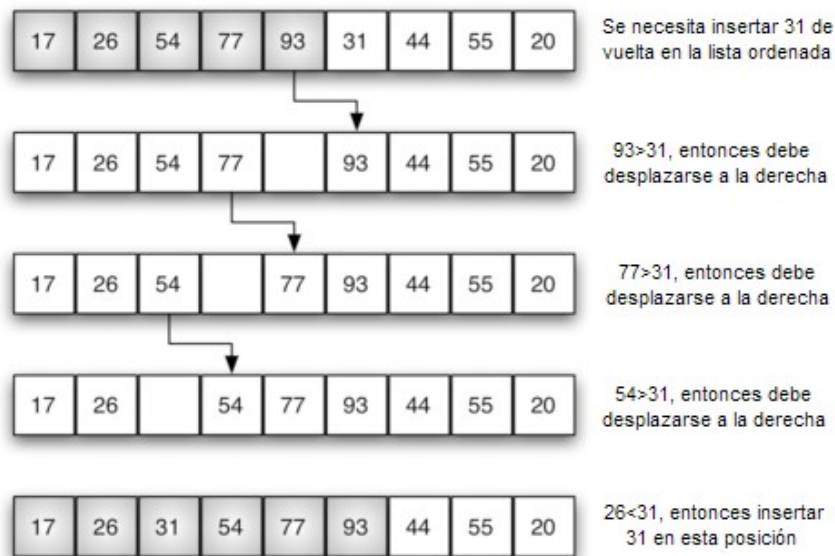
Modificar el algoritmo para ordenar de mayor a menor.



[Vídeo algoritmo selección.](#)

2.3.5 Inserción.

Similar al anterior, pero en este caso no se hace una búsqueda en la parte desordenada sino que se toma el primer elemento de la posición no ordenada y se inserta en la posición correcta en la ordenada.



Ejemplo algoritmo selección.



De los tres algoritmos vistos anteriormente. ¿Cuál elegir?

La implementación en Java con el mismo vector que en los anteriores es:

```
int[] vector = {14, 5, 3, 7, 52, 500, 68, 6, 10, 12, 15, 78, 1};
int indice, temporal;
int indice_ordenado = 0;
//se hace hasta la longitud -1 ya que en el último no se inserta
ninguno
for (int i = indice_ordenado; i < vector.length-1; i++) {
    //se toma el siguiente
    indice = indice_ordenado+1 ;
    //se va de indice mayor a indice menor mientras no se llegue a 0 o
    //el valor del anterior sea menor
    for (int j = indice_ordenado; j >= 0 && vector[indice] < vector[j];
j--) {
        //se realiza el intercambio
        temporal = vector[indice];
        vector[indice] = vector[j];
        vector[j] = temporal;
        //se actualiza el indice del que se tiene que insertar
        indice=j;
    }
    //se incrementa la parte ordenada en 1
    indice_ordenado++;
}
```

El resultado es:

1 3 5 6 7 10 12 14 15 52 68 78 500

Vídeo [algoritmo insercción.](#)

Para saber más: [Algoritmo QuickSort UCAM.](#)

De igual forma que la búsqueda binaria, Java (y muchos otros lenguajes) implementan de forma nativa algoritmos de ordenación más optimizados, los vectores son objetos (se trata en el tema 4) y poseen un método estático que permite ordenar de forma “automática” vectores de elementos:

The screenshot shows the Java API documentation for the `Arrays.sort()` method. The browser address bar shows the URL `https://docs.oracle.com/en/java/javase/12/docs/api/java.base/java/util/Arrays.html`. The documentation lists several static methods for sorting arrays. The method `sort(int[] a)` is highlighted with a red box. The description for this method is "Sorts the specified array into ascending numerical order."

El algoritmo de ordenación que utiliza es el Dual-Pivot Quicksort, y cuya complejidad temporal es $n \cdot \log n$.

Un ejemplo de uso:

```
int[] v={4,5,11,1,2,45,400,3,2};
Arrays.sort(v);
for(int i:v){
    System.out.print(i+",");
}
System.out.println("");
```

La salida del mismo es:

1,2,2,3,4,5,11,45,400



Ejercicios para practicar.

1. Implementar el algoritmo de ordenación de selección y probar con un vector de caracteres.
2. Ahora en vez de utilizar el algoritmo de selección usar el método estático `Array.Sort()`. Comparar los resultados.

2.4. Cadenas de caracteres.

En los primeros años de la informática aparece la necesidad de definir, manejar y mostrar textos, conocidos coloquialmente como **cadenas**, la solución fue representarlas usando vectores o arrays de tipo char o carácter.

La utilización de arrays para representar cadenas conlleva una serie de problemas como son:

- Se ha de calcular de forma aproximada el tamaño de la cadena.
- Se ha de indicar de alguna forma cuando finaliza una cadena dentro del vector.
- Es posible que el tamaño del vector no sea lo suficientemente grande para almacenar la cadena, con lo que se han de realizar operaciones para poder solucionarlo (crear otro array mayor (memoria dinámica) y copiarlo).
- Se han de definir operaciones para el manejo de cadenas, como comparar, concatenar(unir), copiar o conocer la longitud entre otras muchas operaciones.

En C puro, los problemas anteriores los ha de resolver el programador o programadora, aunque existen la librería `string.h` que posee código para solucionar los problemas anteriores.

```
char bula[] = "Bula Fiji"
```

B	u	l	a		F	i	j	i	\0
---	---	---	---	--	---	---	---	---	----

10 elements

```
char bula[15] = "Bula Fiji"
```

B	u	l	a		F	i	j	i	\0					
---	---	---	---	--	---	---	---	---	----	--	--	--	--	--

15 elements

Don't forget that one character is needed to store the *null character* (\0), which indicates the end of the string.

Representación cadena en memoria.

Los lenguajes orientados a objetos manejan cadenas de caracteres para representar los textos, a modo de introducción del próximo tema si se compara con los lenguajes estructurados s:

Estructurados:

- Se define una estructura, por ejemplo un vector de enteros, independiente.
- Se tiene un código independiente que realiza ciertas operaciones sobre las estructuras de ese tipo, por ejemplo invertir(vector)

En los lenguajes orientados a objetos, se unen las estructuras y las operaciones (llamadas métodos) en clases, de forma que al crear una variable de tipo String las operaciones se encuentran "dentro" de esta, siguiendo el ejemplo anterior:

```
Cadena c= new Cadena();  
cadena.invertir();
```

Java posee la clase String, que internamente gestiona un array de cadenas y que implementa una serie de operaciones con ese array y con otros que se les puede facilitar, por ejemplo comparar dos cadenas o concatenar dos cadenas.

La interfaz de programación de aplicaciones o API de la clase String en Java se puede encontrar en:

<https://docs.oracle.com/javase/8/docs/api/java/lang/String.html>

Esta documentación describe las características de la clase y los métodos (operaciones), por ejemplo el método **boolean equals(Object otra_cadena)**, al que se le pasa otra cadena para ver si son iguales, devolviendo un booleano.

equals

```
public boolean equals(Object anObject)
```

Compares this string to the specified object. The result is true if and only if the argument is not null and is a `String` object that represents the same sequence of characters as this object.

Overrides:

`equals` in class `Object`

Parameters:

`anObject` - The object to compare this `String` against

Returns:

true if the given object represents a `String` equivalent to this string, false otherwise

See Also:

`compareTo(String)`, `equalsIgnoreCase(String)`

Un ejemplo de uso:

```
//formas de definir un objeto cadena
String cadena1="Primera cadena";
String cadena2= new String();
cadena2="Segunda cadena";
String cadena3= new String("Tercera cadena");
String cadena4;
cadena4=new String();
//métodos o operaciones.
//concatenación de cadenas "unión"
cadena2.concat(cadena1);
System.out.println(cadena2);
//comparar cadenas, no se ha de usar el ==
System.out.println(cadena2.equals(cadena3));
//obtener la longitud en caracteres de la cadena
int longitud=cadena4.length();
System.out.println("La longitud es "+longitud);
```

Destacar que para acceder a un carácter por posición no se utiliza el operador `[]`, sino métodos concretos, estos son:

```
char cadena.charAt(int indice)
```

Que devuelve el carácter que se encuentra en la posición índice.

Otro método interesante es:

```
int cadena.indexOf(char carácter)
```

Devolviendo la posición de la primera ocurrencia.



En Java las cadenas son inmutables, no se puede modificar, existen otras clases que si permiten modificar los caracteres como `StringBuilder` o `StringBuffer`, con una serie de ventajas e inconvenientes (relacionados con sincronización).

Un ejemplo de uso de estos métodos es dar la vuelta a un texto, en este caso se trabaja con una array de char, con un String no es posible modificar el valor de las celdas:

```
String resultado;

char[] vector_cadena=null;
int longitud;
char temporal;
BufferedReader br;
br = new BufferedReader(new InputStreamReader(System.in));
System.out.println("Introducir la cadena a rotar:");
//es posible que la operación de fallo, se añade control del posible
fallo.
try {
    vector_cadena = br.readLine().toCharArray();
} catch (IOException ex) {
    System.out.println("Error");
}
//se obtiene la longitud de la cadena
longitud = vector_cadena.length;
//se recorre la mitad y se intercambia
for (int i = 0; i < longitud / 2; i++) {
    temporal = vector_cadena[i];
    vector_cadena[i] = vector_cadena[longitud - i - 1];
    vector_cadena[longitud - i - 1] = temporal;
}
//se crea una nueva cadena y se almacena
resultado = new String(vector_cadena);
System.out.println(resultado);
```

2.5. Arrays multidimensionales.

En ocasiones es necesario definir vectores de más de una dimensión, denominándose matrices. El ejemplo más sencillo es una matriz de 2 dimensiones que se puede representar como filas y columnas haciendo el símil con las hojas de cálculo.

El manejo de estas matrices es similar al de una dimensión, con la salvedad que en la creación **se ha de reservar espacio para todas las dimensiones** y a la hora de acceder dependiendo del número de índices se puede acceder a un array de una dimensión menos o a una celda concreta. Por ejemplo:

Crear un array de 2 dimensiones:


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

Si se accede con un único índice:

```
a[1]
```

Se está accediendo a un array de una dimensión y 4 celdas.

Si se accede con dos índices se accede a un número entero.

```
a[1][0]
```

	Column 0	Column 1	Column 2
Row 0	x[0][0]	x[0][1]	x[0][2]
Row 1	x[1][0]	x[1][1]	x[1][2]
Row 2	x[2][0]	x[2][1]	x[2][2]

Array bidimensional

Para definir un array de 3 dimensiones, reservar la memoria e inicializarlo:

```
//declaración  
int [][][] matriz3d;  
//reserva de memoria  
matriz3d= new int[3][3][3];
```

Para darles valores o leer todos los valores:

```
//inicializar  
for (int i = 0; i < 3; i++) {  
    for (int j = 0; j < 3; j++) {  
        for (int k = 0; k < 3; k++) {  
            matriz3d[i][j][k] = k + j * 10 + i * 100;  
        }  
    }  
}  
//mostrar  
for (int i = 0; i < 3; i++) {  
    for (int j = 0; j < 3; j++) {  
        for (int k = 0; k < 3; k++) {  
            System.out.print(matriz3d[i][j][k] + "\t");  
        }  
    }  
}
```

```
        System.out.println(" ");  
    }  
    System.out.println(" ");  
}
```

Es posible declarar, reservar e inicializar al mismo tiempo, al igual que los arrays unidimensionales:

```
//declaracion, reservar de memoria e inicialización  
int[][] matriz2d = {  
    {1, 2, 3},  
    {4, 5, 6}  
};
```



Si se quiere recorrer un array de 2 dimensiones ¿Cuántos bucles son necesarios?
¿Y si tiene 4 dimensiones?.

En el siguiente ejemplo se crea un tablero para jugar al buscaminas, inicialmente se coloca un 20% de minas de forma aleatoria (con la variable double porcentaje), una celda tiene mina si vale 100 y a continuación se calcula el valor de cada celda (el valor de la celda es el número de minas que tiene alrededor).

```
import java.util.Scanner;  
  
/**  
 *  
 * @author Pedro  
 */  
public class Buscaminas {  
  
    public static void main(String argv[]) {  
        int[][] buscaminas;  
        int alto, ancho;  
        int num_minas;  
        double porcentaje = 0.20d;  
        double aleatorio;  
        Scanner input = new Scanner(System.in);  
        //se pide el ancho y el alto  
        System.out.println("Introduce el número de filas");  
        alto = input.nextInt();  
        System.out.println("Introduce el número de columnas");
```

```
ancho = input.nextInt();
//se crea la matriz
buscaminas = new int[alto][ancho];
//se colocan las minas
for (int i = 0; i < alto; i++) {
    for (int j = 0; j < ancho; j++) {
        //devuelve un número entre 0 y 1 aleatoriamente
        aleatorio = Math.random();
        //si esta entre 0 y 0,1 se le pone una mina
        if (aleatorio <= porcentaje) {
            buscaminas[i][j] = 100;
        } else {
            buscaminas[i][j] = 0;
        }
    }
}
//se recorre la matriz para ver si tiene minas cerca, mirando en x,y
/*
celda(x-1,y-1)  celda(x,y-1)      celda(x+1,y+1)
celda(x,y-1)    celda(x,y)        celda(x,y+1)
celda(x+1,y-1)  celda(x,y+1)      celda(x+1,y+1)
*/
for (int i = 0; i < alto; i++) {
    for (int j = 0; j < ancho; j++) {
        //si es menor de 100 entonces no es una mina
        if (buscaminas[i][j] < 100) {
            //se puede mirar en las 8 con código o hacer un bucle
            //se hace con bucle
            for (int k = (i - 1); k <= (i + 1); k++) {
                for (int m = (j - 1); m <= (j + 1); m++) {
                    //no se tienen que mirar lo de los extremos
                    if ((k >= 0 && k < (alto) && m >= 0 && m < (ancho))
&& !(k == i && m == j)) {
                        //si la que se esta mirando es una mina se le
suma uno
                        if (buscaminas[k][m] > 50) {
                            buscaminas[i][j]++;
                        }
                    }
                }
            }
        }
    }
}
```

```

    }

    //se muestra el buscaminas
    for (int i = 0; i < alto; i++) {
        for (int j = 0; j < ancho; j++) {
            System.out.print(buscaminas[i][j] + "\t");
        }
        System.out.println("");
    }
}

```

Con Math.random() se obtiene un double aleatorio entre 0 y 1. Un ejemplo de salida:

```

Introduce el número de filas
5
Introduce el número de columnas
10
0      0      1      1      2      100      2      100      2      1
1      2      3      100      2      1      2      2      100      1
1      100      100      2      1      0      0      1      1      1
1      2      2      1      0      0      0      0      0      0
- 0      0      0      0      0      0      0      0      0      0
-----

```

Ejemplo tablero buscaminas.



Ejercicios para practicar.

- Definir una matriz de 10x20, instanciarla, iniciarla con valores aleatorios entre 0 y 1000 y:
 - Buscar el número mayor e indicar la posición en la que se encuentra.
 - El menor.
 - Obtener la suma de todos los elementos
- Crear un programa que solicite el número de filas y columnas de una matriz de dos dimensiones, cree 2 matrices con valores aleatorios y realice la suma de dichas matrices.

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} + \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} = \\
 = \begin{pmatrix} 1+5 & 2+6 \\ 3+7 & 4+8 \end{pmatrix} = \begin{pmatrix} 6 & 8 \\ 7 & 12 \end{pmatrix}$$

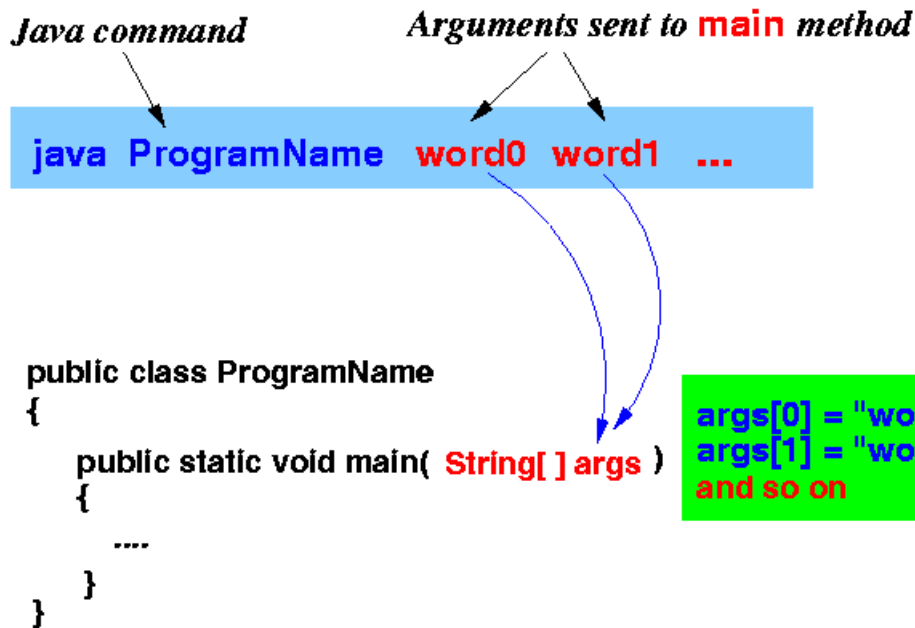
2.5.1 Paso de parámetros a programas.

El punto de entrada de los programas creados es **public static void main (String argv[])**.

Al analizarlo con más detalle se observa que aparece la declaración:

```
String argv[]
```

Es un array de cadenas en el que se almacenan los parámetros introducidos desde la línea de comandos un vector de cadenas.



Vector de cadenas en programa.

El siguiente ejemplo se ha extraído de la documentación oficial:

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

Y ante la entrada:

```
java Echo Drink Hot Java
```

La salida es:

```
Drink  
Hot  
Java
```

Es posible transformar las cadenas a otros tipos como int, float o boolean:

```
int firstArg;  
if (args.length > 0) {  
    firstArg = Integer.parseInt(args[0]);  
}
```

Observar como en primer lugar se comprueba si el número de argumentos es mayor de 0 para comprobar que se le ha pasado alguno, para a continuación realizar la conversión.

2.5.2 Conversiones.

Los tipos primitivos tienen sus equivalentes en clases, el tipo `int` tiene la clase `Int`, el tipo `float` la clase `Float`... Estas clases poseen métodos estáticos (código) con muchas funcionalidades entre ellas la conversión entre tipos. **Los métodos estáticos se verán en temas posteriores, pero son fragmentos de código asociados al tipo de dato(clase) que pueden ser utilizado sin necesidad de crear el objeto, simplemente:**

```
nombreClase.método_a_usar.
```

Todas estas clases poseen un método para convertir un texto al tipo de dato:

```
int Integer.parseInt(String cadena);  
float Float.parseFloat(String cadena);  
double Double.parseDouble(String cadena);
```

Se pueden consultar la documentación de cada clase en la página de Oracles, por ejemplo para `Float`:

<https://docs.oracle.com/javase/8/docs/api/java/lang/Float.html>



¿Y para pasar tipos primitivos a cadenas? Indagar en la documentación (API) de anteriormente mencionadas.



Ejercicios para practicar.

1. Escribir un programa que indique el número de parámetros que se le pasa por el terminal.
2. Implementar un algoritmo de ordenación de los vistos en clase que ordene las cadenas que se le pasan como parámetros a un programa desde el terminal.

2.6. Estructuras.

Los vectores son un concepto muy útil pero no son los suficientemente expresivos en ciertas situaciones, pensar en la gestión de coordenadas de un programa GPS, la ficha de un alumno o la representación de una factura.

En caso de administrar una ruta GPS, en la que se tiene longitud, latitud y altura usando vectores de tipos primitivos es necesario gestionar 3 vectores, uno para la longitud, otro para la latitud y otro para la altura. Al tener los 3 vectores, las operaciones se vuelven más complejas, los fallos más usuales y el desarrollo del código es poco natural.

La solución es agrupar en una única unidad los 3 valores que identifican un punto, la latitud, longitud y altura, denominándose a esta agrupación estructura.

Una estructura además se puede ver como un nuevo tipo de dato (en C se pueden y deben definir nuevos tipos de datos).



Una estructura se identifica con un nombre y está compuesto de un conjunto de tipos de datos que pueden ser primitivo, vectores, arrays u otras estructuras identificadas por un nombre dentro de la estructura.

Ejemplos de definición de estructuras algunos tipos de datos:

En C/C++:

```
struct [structure tag] {  
  
    member definition;  
    member definition;  
    ...  
    member definition;  
} [one or more structure variables];
```

```
struct Books {  
    char title[50];  
    char author[50];  
    char subject[100];  
    int book_id;  
} book;
```

En PHP:

```
struct StructName {  
    fieldType1 $field1;  
    fieldType2 $field2, $field3;  
}
```

```
struct Employee {  
    string $firstName;  
    string $lastName;  
    int $salary;  
    bool $fullTime;  
}
```

En C#:


```
public struct Coords
{
    public Coords(double x, double y)
    {
        X = x;
        Y = y;
    }

    public double X { get; }
    public double Y { get; }

    public override string ToString() => $"({X}, {Y})";
}
```

Una vez definida la estructura es posible declarar variables de tipo de dato la estructura definida previamente y trabajar como una variable más. Un ejemplo en C:

```
//declaración de la estructura.
struct Person {
    char name[50];
    int citNo;
    float salary;
};
```

Declaración de variables e instanciación (en C cuando se define la variable se reserva la memoria, excepto con punteros, pero esto queda fuera del curso).

```
struct Person person1, person2,
```

Además se pueden dar valores a los elementos internos al mismo tiempo que se declara .

```
struct Person person3={"Luis Mis",3,27656.0};
```



Para acceder a los miembros de la variable de tipo estructura se usa el operador punto .

```
strcpy( person1.name, "Antonio Tonio\0");
printf("El nombre de Person 1 es %s\n",person1.name);
printf("El nombre es %s\n",person3.name);
printf("El citNo es %d\n",person3.citNo);
printf("El sueldo es %f\n",person3.salary);
```

El ejemplo completo:

```
#include <stdio.h>
#include <string.h>
struct Person
{
    char name[50];
    int citNo;
```

```
float salary;
};
int main () {
    struct Person person1, person2;
    struct Person person3 = { "Luis Mis", 3, 27656.0 };

    strcpy (person1.name, "Antonio Tonio\0");
    printf ("El nombre de Person 1 es %s\n", person1.name);
    printf ("El nombre es %s\n", person3.name);
    printf ("El citNo es %d\n", person3.citNo);
    printf ("El sueldo es %f\n", person3.salary);

    return 0;
}
```



Comparar cómo se asignan cadenas en Java y en C.

Por desgracia Java no posee estructuras como tal y se han de utilizar clases (una clase se puede ver como datos (estructura) más operaciones, en caso de no tener operaciones (se verá en el tema 4), una clase se puede comportar como una estructura de otros lenguajes).

En este punto se explica como hacer que una clase en Java se comporte como una estructura, por supuesto una clase es mucho más expresiva que una estructura pero dado que el concepto existe en muchos lenguajes es importante conocer el funcionamiento y la filosofía de las estructuras.

Para crear una clase en Java se utiliza la palabra reservada **class** a continuación en **nombre de la clase** (se recomienda primer carácter en mayúsculas de forma que se pueda diferenciar de nombre de variables, en minúsculas, se inicia el cuerpo abriendo llave, declarando en el mismo los atributos (las variables dentro de una clase se denomina **atributo**), cada atributo tiene el tipo de dato/clase que la define y un nombre (en minúsculas), por último se cierran las llaves.

Para crear una variable con esa clase que simula una estructura se declara la variable de ese tipo y **a continuación se ha de inicializar/reservar memoria**.

```
//se define la clase persona, que simula una estructura en Java
class Persona {
```

```
String nombre;  
int numero;  
float sueldo;  
}  
//clase principal  
public class Main {  
    public static void main(String[] args) {  
        Persona personal;  
        //se ha de reservar memoria/instanciar  
        personal= new Persona();  
        //observar el operador punto .  
        personal.nombre="Manolo Tono";  
        personal.numero=33;  
        personal.sueldo=21789.4f;  
        System.out.println("El nombre es "+personal.nombre);  
    }  
}
```



Pensar como definir una estructura para almacenar la información de una coordenada de GPS (altitud,latitud y longitud). Definirlas en C y en Java. ¿Y si se desea gestionar los terrenos de una localidad? Se necesitan las dimensiones (asumir cuadradas) y la localización.



Ejercicios para practicar.

1. Definir una estructura para representar puntos en 2D (reales), crear 2 variables de ese tipo.

2.7. Estructuras y vectores.

Los vectores son conjuntos ordenados de elementos del mismo tipo de dato, este tipo de dato puede ser tipos de datos primitivos, estructuras o clases.

En el caso de estructuras y/o clases la forma de operar con vectores o arrays es exactamente igual, pero con la salvedad de que **es necesario instanciar cada elemento de forma individual**.

```
class Coordenada{
```

```
float longitud;  
float latitud;  
float altura;  
}  
//clase principal  
public class Main {  
  
    public static void main(String[] args) {  
        Coordenada[] ruta= new Coordenada[10];  
        //ESTE CÓDIGO DA FALLO, ES NECESARIO CREAR LAS 10 COORDENADAS  
        // ruta[0].longitud=10.0f;  
        for(int i=0;i<ruta.length;i++){  
            ruta[i]=new Coordenada();  
        }  
        //AHORA SI ES POSIBLE ACCEDER  
        ruta[0].longitud=10.0f;  
    }  
}
```



¿Se quiere tener una representación en 3d de una montaña? ¿Cómo inicializar la representación seleccionada? Indicar como acceder en la posición central a su altitud.



Ejercicios para practicar.

1. Crear un vector que permita gestionar una figura (polígono) de como máximo 10 puntos, se ha de poder modificar un punto concreto y obtener los valores indicados por el índice.

3. Actividades y ejercicios.

- ¿Para qué se utilizan los enumerados? ¿Qué ventajas ofrecen?
- Indicar las operaciones principales que se definen con los arrays. Explicarlas sobre un array de Booleanos.
- Definir los conceptos de declara, instanciar e inicializar en los vectores.
- Los vectores en memoria se almacena de forma _____
- En caso de tener un vector ordenado. ¿Qué algoritmo de búsqueda usar?

? Se tiene un vector ordenado de 100 elementos, comparar el uso de búsqueda binaria y secuencial. Número máximo de iteraciones en ambos casos.

? De los 3 algoritmos básicos de ordenación. ¿Cuál es el peor?

? ¿Cómo se representan los textos en los lenguajes de programación?

? ¿Qué problemas se han de manejar al usar cadenas con vectores?

? ¿Basta con definir una variable de tipo String en Java para usarla? ¿Qué es necesario?

? Buscar y explicar si existen los siguientes métodos en la clase String de Java:

- Borrar espacios en blanco.
- Pasar a mayúsculas.
- Pasar a minúsculas.
- Comparar cadenas iguales.
- Trocear cadenas, por ejemplo si se tiene la cadena "Hola Mundo" ha de devolver ["Hola", "Mundo"]

? Pasar una cadena a Booleano. Este método tiene algo especial, indicar el qué

? ¿Cuántas dimensiones puede una matriz?

? Si se desea inicializar una matriz de 4 dimensiones ¿Cuántos bucles usar?

? Razonar la siguiente afirmación: En una matriz se pueden tener valores de diferentes tipos.

? Se tiene la matriz:

```
int a[][][] []=new int[10][20][30][40];
```

Indicar a qué se está accediendo al hacer:

- a[10];
- a[7][;]
- a[7][2];
- a[7][2][1][3];

? Explicar cómo pasar parámetros a un programa en Java desde la línea de comandos. Escribir en Java o pseudocódigo un algoritmo que compruebe que se le pasan 4 parámetros en pares, a un programa de la siguiente forma -peso 78 -altura 185, el orden de peso y altura pueden ir en diferente orden.

? .Definir que es una estructura.

? Diferencias entre estructuras en C y Java.

? Definir una estructura para almacenar los datos de una vivienda.

? ¿Este código en Java funciona?Indicar la razón. En caso de que no funcione solucionarlo.

```
Vivienda v;  
v.precio=100000;
```

? ¿Qué operador se utiliza para acceder a los miembros de una estructura?

? Se desea gestionar 100 viviendas a partir de la estructura definida en ejercicios anteriores. Explicar cómo realizarlo.

? Explicar como aplicar el algoritmo de ordenación de inserción al ejercicio de las 100 vivienda ordenando por precio.

? Implementar el algoritmo de búsqueda binaria en las 100 viviendas.