# Estructura del Array

* Un ``Array'' es una lista de elementos del mismo tipo (homogénea), cada uno ellos identificado por un índice.
* Se puede acceder a los elementos del Array (para guardar o recuperar la información) en cualquier orden.
* El Array queda definido por
  + el tipo básico de elementos que aglutina,
  + el número de elementos (longitud).

\begin{figure}\input PIIt2cas1.sty
\end{figure}

* Es la estructura de datos que se usa más frecuentemente en programación.
* La gestión de información con el Array es eficiente. De hecho, la memoria del ordenador se gestiona como un Array en los lenguajes de bajo nivel.
* Los Arrays se utilizan con tanta frecuencia que todos los lenguajes de programación ofrecen algún modo para utilizarlos con facilidad.

# Arrays en Java

* En Java, los Array pueden ser de cualquier tipo de dato, incluidos objetos.
* El tipo de dato Array es, a su vez, un objeto.
* Las variables del tipo Array se declaran utilizando **[]**, del siguiente modo:

tipo\_basico[] nombre\_variable

* Por ejemplo,

int[] fila;

Declara la variable fila del tipo Array de datos del tipo int.

* Estas variables almacenarán la referencia al objeto.
* Para crear el objeto, se utiliza el operador **new** de la forma:

new tipo\_basico[numero\_de\_elementos]

* Por ejemplo,

fila = new int[100];

Crea el objeto que puede almacenar 100 enteros.

* Se puede declarar la variable y crear el objeto en la misma instrucción:

int[] fila = new int[100];

* A cada uno de los elementos del Array se accede con el nombre seguido del índice entre corchetes. El índice varía entre 0 y el número de elementos que puede almacenar. Así,

fila[99] = 1763;

almacena el valor 1763 en la última posición del Array fila.

* Cada uno de los componentes se puede utilizar como una variable cualquiera del tipo de dato básico que almacena.
* Por ejemplo, el Array fila ofrece la 100 variables del tipo int:

fila[0], fila[1], ... ,fila[98], fila[99]

* El objeto Array incluye una variable que es la longitud y que tiene el nombre **length**. La variable es pública y se puede acceder a ella directamente como un campo del objeto.

En el ejemplo de fila, este programa imprime 100:

public class p2t2p1{

public static void main(String[] args)

{ int[] fila = new int[100];

int cuantos = fila.length;

System.out.println(cuantos);} }

 El esquema del objeto Array es:

\begin{figure}\input PIIt2cas2.sty
\end{figure}

 La variable length es del tipo final, se asigna en la creación del objeto y no se puede modificar posteriormente.

 Cuando se crea el objeto, todos los elementos toman un valor inicial. Si se trata de tipos b

 Como ejemplo, vamos a estudiar el uso de un Array en un programa para calcular la tabla de **Eratóstenes**.

 La tabla, que ha de contener todos los números primos (hasta un determinado valor) se construye con el algoritmo de la criba (también debido a Eratóstenes).

 Se crea una tabla con todos los números, y de ella se van eliminando los números compuestos: los que resultan de los productos de los primos.

\begin{figure}\input PIIt2cas3.sty
\end{figure}

* Si se quiere calcular los primos hasta el número MAX, el algoritmo de la criba sería:

Para i desde 2 hasta MAX/2

Para j entre 2 y MAX/i

tachar los múltiplos de la forma i\*j

* Para organizar los números de la tabla podríamos utilizar un Array con todos los números en la tabla y la operación de ``tachar'' podría ser eliminarlos de la tabla.
* Pero no es necesario. La información de cuál es el número puede estar contenida en el índice de modo que lo único que tiene que almacenarse en la tabla es si está ``tachado'' o no.
* Una buena elección será utilizar un Array de datos boolean con el valor true si es primo y false si es compuesto (está ``tachado'').
* A qué número corresponde la propiedad, estará en el índice de la tabla asociado al valor.
* Dado que los índices empiezan por 0, la propiedad del número i se almacenará en la posición i -1 del Array.
* El programa que crea la tabla podría ser:

# La criba de Eratóstenes

public class p2t2p5{

public static void main(String[] args){

// Cálculo de los numeros primos entre 1 y MAX

// utilizando el algoritmo de Eratostenes

int i,j; final int MAX = 70;

boolean[] esPrimo = new boolean[MAX];

// Se inicializa la tabla

esPrimo[0] = false;

for (i = 0; i< MAX;i++) esPrimo[i] = true;

// Empieza la criba

for(i = 2 ; i <= (MAX/2);i++)

for(j = 2; j<=(MAX/i); j++) esPrimo[i\*j - 1] = false;

// Se cuenta el número de primos encontrados

j = 0; for(i=1;i<MAX;i++) if (esPrimo[i]) j++;

System.out.println("Hasta el "+MAX+" hay "+j+" primos");

for(i=1;i<MAX;i++) if (esPrimo[i])

System.out.print(i+1+" ");}}

con la salida:

Hasta el 70 hay 19 primos

2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67

### Matrices o Arrays de varios subindices

Podremos declarar arrays de varios subíndices, pudiendo tener arrays de dos niveles, que serían similares a las matrices, arrays de tres niveles, que serían como cubos y así sucesivamente, si bien a partir del tercer nivel se pierde la perspectiva geométrica.

Para declarar e inicializar un array de varios subíndices lo haremos de la siguiente manera:

1. tipo\_dato nombre\_array[][];
2. nombre\_array = **new** tipo\_dato[tamaño][tamaño];

De esta forma podemos declarar una matriz de 2x2 de la siguiente forma:

1. int matriz[][];
2. matriz = **new** int[2][2];

El acceso se realiza de la misma forma que antes:

1. int x = matriz[1][1]; *// Para leer el contenido de un elemento*
2. matriz[1][1] = x; *// Para asignar un valor.*

Hay que tener en cuenta que para mostrar su contenido tendremos que utilizar dos bucles. Para saber el número de columnas lo haremos igual que antes mediante la variable length, pero para saber el numero de filas que contiene cada columna lo tendremos que realizar de la siguiente manera:

1. matriz[numero\_elemento].lenght;

Nuestra lectura de los elementos de una matriz quedaría de la siguiente forma:

1. int matriz[][];
2. matriz = **new** int[4][4];
3. for (int x=0; x < matrix.length; x++) {
4. for (int y=0; y < matriz[x].length; y++) {
5. [**System**](http://www.google.com/search?hl=en&q=allinurl%3ASystem+java.sun.com&btnI=I%27m%20Feeling%20Lucky).out.println (matriz[x][y]);
6. }
7. }
8. El número de filas está almacenado en mat.length y el número de columnas de la fila $k$está almacenada en mat[k].length
9. La estructura interna de un Array bidimensional es:

\begin{figure}\input PIIt2cas4.sty
\end{figure}

# Copias de Arrays

* La copia de los elementos de una Array, se realiza tan frecuentemente que existe un método para ello en la clase **System**.
* El método
* System.arraycopy(afu,ifu,ade,ide,inu)

hace la copia. En los argumentos, afu es el Array con los datos de origen, ade el Array a donde se copiarán los datos, ifu el índice inicial para empezar a copiar elementos en el array original, ide el índice inicial donde empiezan a copiarse los elementos en el destino, e inu el número de elementos que se han copiar.

* Para copiar todos los elementos de un Array a otro, por ejemplo los elementos de a al array b, la llamada al método sería:
* System.arraycopy(a,0,b,0,a.length);
* El array destino tiene que haberse creado con un tamaño suficiente para almacenar todos los elementos que se quieren copiar.
* En el siguiente ejemplo se utiliza este método para copiar integramente un Array, y se puede comprobar la diferencia entre la copia de los elementos y la asignación de los objetos.
* public class p2t2p9{
* public static void main(String[] args){
* int[] a = {12,14,16,18,20,22,24,26};
* int[] b = new int[a.length];
* System.arraycopy(a,0,b,0,a.length);
* for(int i=0;i<b.length;i++) System.out.print(b[i]+
* " "); System.out.println();
* a[1] = 3003;
* for(int i=0;i<b.length;i++) System.out.print(b[i]+
* " "); System.out.println();
* b = a;
* a[1] = 3003;
* for(int i=0;i<b.length;i++) System.out.print(b[i]+
* " ");}}
* y la salida es:
* 12 14 16 18 20 22 24 26
* 12 14 16 18 20 22 24 26
* 12 3003 16 18 20 22 24 26

# Cambio de tamaño

* Si un Array se nos queda pequeño, hay que crear uno nuevo de mayor dimensión y copiar los elementos.
* En el siguiente ejemplo se utiliza un método para aumentar de tamaño un Array de datos del tipo int:
* public class p2t2p11{
* public static int[]
* aumentaIntArray(int[] ant, int cant){
* int[] nue = new int[ant.length + cant];
* for(int i = 0;i< ant.length; i++) nue[i] = ant[i];
* return (nue);}
* public static void main(String[] args){
* int[] a = {11,12,13}; int i;
* for(i=0;i<a.length;i++) System.out.print(a[i]+" ");
* System.out.println();
* a = aumentaIntArray(a,3);
* a[3] = 14; a[4] = 15; a[5] = 16;
* for(i=0;i<a.length;i++) System.out.print(a[i]+" ");
* System.out.println(); }}

y la salida es:

11 12 13

11 12 13 14 15 16

**Oredenar arrays**

A la hora de ordenar un array, podemos perder el tiempo programando una función que haga el trabajo, o bien utilizar la función que nos da ya creada y que la podemos encontrar en java.util.Arrays.  
  
Ahí nos podemos encontrar varios métodos que nos permitirá ordenar arrays de distintos tipos de datos, ya sean enteros, cadenas, objetos....  
  
Vamos a ver un ejemplo de ordenación de un vector de cadena, primero lo veremos ordenado ascendente y luego descendente.  
  
El vector desordenado será el siguiente:

String [] stringArray = {"ab", "aB", "c", "0", "2", "1Ad", "a10"};

Si ahora le aplicamos el método sort de la clase que hemos indicado, el array saldrá ordenado de forma ascendente. El código sería el siguiente:

//Ordacion ascendente

Arrays.sort(stringArray);

System.out.println("\*\*\*\*\*\* Ordenando String Array \*\*\*\*\*\*\*");

for (String str : stringArray) {

System.out.println(str);

}

Si lo queremos ordenar de forma descendente, habría que indicarle al método que lo haga así. Para este caso, el método quedaría de la siguiente forma.

//Ordacion descendente

Arrays.sort(stringArray, Collections.reverseOrder());

System.out.println("\*\*\*\*\*\* Ordenando String Array \*\*\*\*\*\*\*");

for (String str : stringArray) {

System.out.println(str);

}

Otra opción que nos permite el método “sort” es una ordenación selectiva, es decir, de un array, ordenar los elementos que le indiquemos en un rango dado. Es decir, el método en este caso sería de la siguiente forma:

Arrays.sort (Object [], int startIndex, int endIndex)

Y un ejemplo del código sería el siguiente:

//Ordenamos los elementos que van desde la posición 3 a la 6

Arrays.sort(stringArray, 3, 6);

System.out.println("\*\*\*\*\*\* Ordenando String Array \*\*\*\*\*\*\*");

for (String str : stringArray) {

System.out.println(str);

}

# Clase vector

Paquete java.util

El array ([]) es la forma más eficiente que Java proporciona para almacenar y acceder a una secuencia de objetos. El array es una simple secuencia lineal, que hace que el acceso a los elementos sea muy rápido, pero el precio que hay que pagar por esta velocidad es que cuando se crea un array su tamaño es fijado y no se puede cambiar a lo largo de la vida del objeto. Se puede sugerir la creación de un array de tamaño determinado y luego, ya en tiempo de ejecución, crear otro más grande, mover todos los objetos al nuevo y borrar el antiguo. Esto es lo que hacen las **clases Vector y ArrayList**, que se verán posteriormente, pero debido a la carga que supone esta flexibilidad, un Vector y un ArrayList son menos eficiente que un array, en cuestiones de velocidad.

Los elementos de un **vector** pueden ser de cualquier tipo de objeto, incluidos otros vectores o estructuras de datos complejas. En un mismo vector podemos guardar distintos tipos de datos. Siempre podremos recuperarlas como Object. Si guardamos datos de tipo primitivo los aceptará pero luego los tenemos que recuperar como Object es decir como objetos de las clases Integer, Double, etc

Tiene capacidad dinámica, se amplia o se reduce dependiendo de las necesidades del momento.

**Vector vec = new Vector(**); Crea un vector vacío con un tamaño de 0 y una Capacidad de 10 elementos. Cuando meta el elemento 11, la capacidad se duplica a 20.

**Vector vec = new Vector(3)**; crea un vector vacío con una capacidad de 3 elementos, si guardamos el 4, la capacidad se duplica a 6.

**Vector vec = new Vector(6, 4)** Tenemos un vector con un tamaño inicial 0 elementos. Y una capacidad de 6 elementos. Si rebasamos dicha dimensión y guardamos el elemento 7 la capacidad del vector será de 10 no de 12.

**addElement(**valor) añade el valor como último elemento del vector, e incrementa en uno el tamaño del vector.

addElement (new String (“lunes”));

addElement( new Persona(............));

## add(valor) Añade el valor en la última posición.

**add**(indice, valor) Añade el valor en la posición indicada, desplazando los demás elementos.

**size**() devuelve el número de elementos del vector en ese momento, tamaño.

**capacity**() devuelve el número de elementos para los que tiene capacidad el vector, en ese momento.

**trimToSize();** iguala la capacidad del vector al tamaño. Si aumentamos el tamaño se vuelve a duplicar la capacidad,

Ejemplo:

public static void main(String[] args) {

Vector tab=new Vector();

Vector tab1=new Vector(5);

Vector tab2=new Vector(3,5);

System.out.println("capacidad"+tab.capacity()+"tamaño "+tab.size()); //visualizamos capacidad 10 tamaño 0

System.out.println(tab1.capacity()+"tamaño "+tab1.size());

// capacidad 5 tamaño 0

System.out.println(tab2.capacity()+"tamaño "+tab2.size());

// capacidad 3 tamaño 0

tab.addElement("primero");

System.out.println(tab.capacity()+"tamaño "+tab.size());

// capacidad 10, tamaño 1

tab1.add("primero");

System.out.println(tab1.capacity()+"tamaño "+tab1.size());//capacidad 5, tamaño 1

tab2.add(0,"primero");

System.out.println(tab2.capacity()+"tamaño "+tab2.size());

// capacidad 3, 1

for (int i=0;i<10;i++)

tab.addElement("ss");

System.out.println(tab.capacity()+"tamaño "+tab.size());

//capacidad 20 tamaño 11

for (int i=0;i<5;i++)

tab1.addElement("ss");

System.out.println(tab1.capacity()+"tamaño "+tab1.size()); //capacidad 10 tamaño 6

for (int i=0;i<5;i++)

tab2.add("ss");

System.out.println(tab2.capacity()+"tamaño "+tab2.size()); //capacidad 8 tamaño 6

tab.trimToSize(); // igualamos tamaño y capacidad

System.out.println(tab.capacity()+"tamaño "+tab.size());

//capacidad 11 tamaño 11

tab.add("ww");

System.out.println(tab.capacity()+"tamaño "+tab.size());

//capacidad 22 tamaño 12

**addAll(**colección) añade todos los miembros de la colección.

**elementoAt (**posicion**)** devuelve el valor del elemento situado en esa posición, como tipo Object, tenemos que convertirlo al tipo de datos deseado.

**Articulo art = (Articulo) obj.elementAt(i);**

Si no sabemos el tipo del elemento**:**

**Class obj = elementoAt(**posicion)**.getClass();**

**String nombreclase= obj.getName();**

**get(**posición) devuelve el elemento situado en ese posición

Persona nueva = new Persona();

Nueva = (Persona) vec.get(3);

**Object obj = set(posición, objeto)** Machaca el objeto de la posición sustituyéndolo por el nuevo. Me devuelve el objeto que estaba en dicha posición.

**firstElement()** obtenemos una referencia al primer elemento del vector. Devuelve el primer elemento del vector.

**LastElements()** Devuelve el último elemento.

**lastIndexOf(**elemento**)** devuelve el índice de la última ocurrencia del elemento.

**contains(**elemento) devuelve TRUE si el elemento está en el vector

**indexOf**(elemento) devuelve el índice donde esta el elemento, la primera ocurrencia.

**toArray()** genera una matriz de Object con los elementos del vector en orden.

**copyInto(**matriz) copia los componentes del vector en la matriz

**remove(**indice**)** elimina el elemento de esa posición.

**removeAllElements()** borra todos los elementos.

**remove**(elemento) borra el elemento, la primera ocurrencia.

**Clear()**  elimina todos los elementos del vector.

**Clone()** obtiene una copia del vector.

**setSize** (int newSize) Define el tamaño de un vector.

**toString** () Devuelve una representación de tipo cadena para cada elemento del vector. Si el vector tiene 10 elementos y en cada uno de ellos tiene “hola”, obtendríamos:

[hola, hola, hola, hola, hola, hola, hola, hola, hola, hola]

**elements()** Devuelve una referencia del tipo enumeración**.**

Enumeration es una interface implementada por la clase vector que dispone de dos métodos.

**hasMoreElements()** devuelve true si hay más elementos

En la enumeración.

**NextElements()** devuelve una referencia al siguiente elemento de la Enumeración.

Ejemplo: Enumeration l = vec.elements();

While(l.hasMoreElements() ==true)

System.out.println(l.nextElements();

**Clase ArrayList**

paquete java.util .

Un ArrayList contiene tantos objetos como necesitemos. No es de tamaño fijo.

Los elementos de un **ArrayList** pueden ser de cualquier tipo de objeto, incluidos otros vectores o estructuras de datos complejas. En un mismo ArrayList podemos guardar distintos tipos de datos. Siempre podremos recuperarlas como Object. Si guardamos datos de tipo primitivo los aceptará pero luego los tenemos que recuperar como Object es decir como objetos de las clases Integer, Double, etc

ArrayList tiene varios constructores, dependiendo de cómo necesitemos construir el ArrayList .

**ArrayList()** construye un ArrayList con capacidad cero por defecto, pero crecerá según le vayamos añadiendo:

ArrayList al = new ArrayList();

**ArrayList(int capacidadInicial**) construye un ArrayList vacío con una capacidad inicial especificada:   
ArrayList al2 = new ArrayList(5);

**Métodos**

**add(objeto) :** añade un objeto al final del arrayList

**add(posición, objeto)** inserta el objeto en la posición desplazando los demás elementos

**Object objeto=get(posición)** nos devuelve el objeto que hay en esa posición. Siempre de tipo Object.

**set(posición, objeto)** sustituye el elemento de la posición por el objeto nuevo.

**remove(posición)** borra el objeto situado en la posición posición.

**remove**(elemento) borra el elemento , la primera ocurrencia.

**clear()** borra todos los objetos del arrayList

**size()** devuelve el tamaño del arrayList

**lastIndexOf(**elemento**)** devuelve el índice de la última ocurrencia del elemento. Si no está devuelve -1.

**contains(**elemento) devuelve TRUE si el elemento está en el vector

**indexOf**(elemento) devuelve el índice donde está el elemento, la primera ocurrencia. Si no está devuelve -1.

**toString** () Devuelve una representación de tipo cadena para cada elemento del vector. Si el vector tiene 10 elementos y en cada uno de ellos tiene “hola”, obtendríamos:

[hola, hola, hola, hola, hola, hola, hola, hola, hola, hola]

**Diferencia entre Vector y ArrayList**

Los vectores crecen duplicando su espacio si no le damos una capacidad de incremento.  
Los ArrayList crecen añadiendo el 50% de su espacio anterior.

Sincronización: La clase Vector es sincronizada (synchronized), por tanto, su contenido está protegido de otros hilos o proyectos, al contrario, los ArrayList no son sincronizados, Hay que tener en cuanta esto porque los vectores tienen un coste en tiempo de ejecución que no tienen los ArrayList pero en cambio son más seguros.