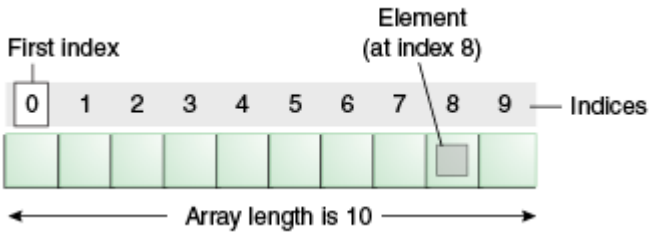
ARRAYS

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/arrays.html

Después de echar un vistazo rápido al enlace anterior podemos decir:



- Un array es un trozo de memoria contiguo(sin fragmentar)
- Contiene una colección de elementos. Podemos ver cada elemento del array como una variable y entonces un array es desde este punto de vista una colección de variables. por ejemplo Si al array del dibujo de arriba se llama miArray tengo las variables miArray[0], miArray[1],...miArray[9]. Es decir el nombre de cada variable o posición consiste en concatenar el nombre del array con su posición indicada como un índice.
- En java todos las variables de un array son obligatoriamente del mismo tipo, por ejemplo todas de tipo int
- En dichas variables se pueden leer y escribir. Para acceder a cada posición(variable), aprovechando que se almacenan una tras otra en el espacio contiguo, se usa un sistema de índices por ejemplo si el array se llama *miArray* y es un array de int
    - para meter en la primera posicion el valor 3
      miArray[0]=3
    - para leer el valor de la segunda posición e imprimirlo por pantalla
      System.out.println(miArray[1])

Digamos que cada variable de un array tiene un nombre ya automáticamente asignado que consiste en *nombreArray[indice]*

UN ARRAY ES UN OBJETO.

El término *array* se traduce por "arreglo". No se utiliza mucho esta traducción. Sí que se utilizan habitualmente como sinónimos de *array* los términos "matriz" y "tabla", aunque ambos términos se suelen referir específicamente a un array de dos dimensiones. Si el array es de una dimensión podemos encontrarnos con el término específico de "vector".
En java los arrays son objetos (no es un tipo primitivo), y por tanto, se crean como cualquier otro objeto, con el operador new. No obstante, al ser un objeto tan usado tiene en su uso cuestiones de sintaxis específicas.

IMPORTANTE: como ya indicamos para las referencias de objetos , a menudo se nombra al objeto con el nombre de una referencia que lo señala. Por ejemplo, para:

```
Coche x = new Coche();
```

Decimos coloquialmente para el caso anterior que, *x es un coche*, salvo que convenga aclarar que, *x es una referencia a un coche*. De la misma forma, para:

```
int[] x= new int[10];
```

decimos que *x es un array de 10 enteros*, salvo que por el contexto convenga aclarar que *x es una referencia a un array de 10 enteros*.

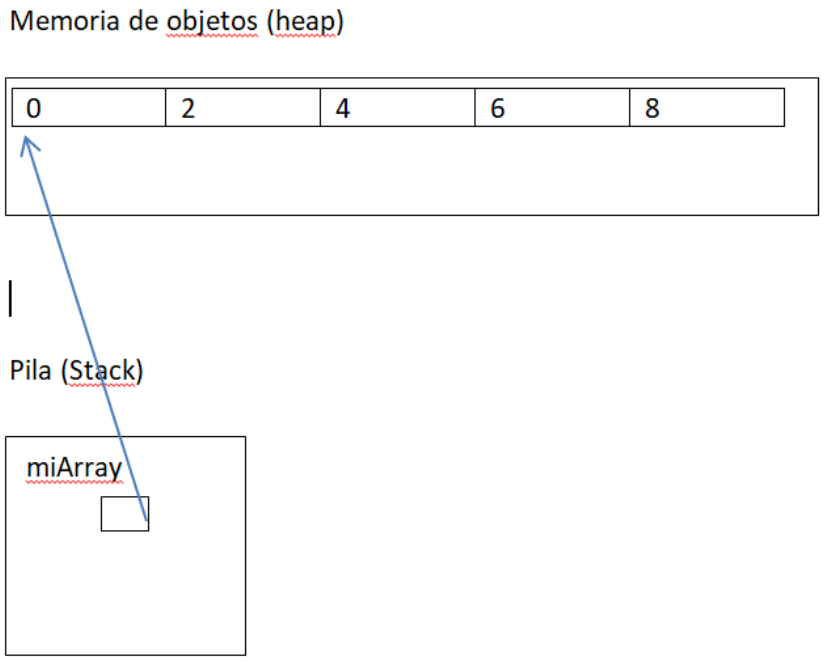
ACCESO A LOS ELEMENTOS DE UN ARRAY CON nombreDelArray[pos]

Ejemplo: Ejecuta el siguiente código

```
class Unidad4{
  public static void main(String[]  args){
    int[] miArray= new int[5];

    //escribimos en el array
    for(int i=0;i<5;i++)
     miArray[i] =i*2;

    //leemos del array
    for(int i=0;i<5;i++)
     System.out.println("miArray[" + i +"]:"  + miArray[i]);
  }
}
```

Observa: creamos un array de 5 elementos. Sus elementos son miArray[0], miArray[1],miArray[2],miArray[3], miArray[4].

Para el ejemplo anterior nos podemos imaginar la siguiente situación en memoria:



La variable miArray es una referencia, es decir, una variable que almacena una dirección de memoria (una referencia) que "apunta" al verdadero array de enteros.

Observa que miArray =>almacena la dirección del principio del array
los []=> son el equivalente la operador . en los otros objetos, es decir indican un desplazamiento desde el principio del array
miArray[3] podría verlo equivalente a *miArray.elementocuartodelarray*

UNA VEZ MÁS Y MUY IMPORTANTE: no confundir variable referencia con objeto
Recuerda que
Coche c = new Coche();
lo podíamos escribir en dos instrucciones
Coche c;
c=new Coche();
De esta segunda forma, menos compacta, queda bien claro que una cosa es la variable referencia y otra el objeto.
Un array es un objeto así que hay las mismas consideraciones, simplemente hay que adaptarlas a la sintaxis especial arrays
int[] x = new int[5];
se puede escribir en dos instrucciones:
int[] x;
x=new int[5];

observa y entiende que no se puede indicar un tamaño al declarar una referencia Por ejemplo:
int[5] x; ¡no tiene sentido!
ya que de una variable referencia lo único que hay que saber es su tipo en este caso que va a referenciar a un array de enteros. Las referencias almacenan una referencia de memoria a un objeto, lo que tienen un tamaño de enteros son los objetos arrays no las referencias a arrays.

int[] x;
x=new int[5];
x=new int[6];
x siempre tiene que referenciar a un array de int (o a null), "el tamaño no importa"

Ejercicio U4_B11_E1: Crea un array que almacene las 5 vocales y luego las imprime.

Ejercicio U4_B11_E2: Añade el código necesario al siguiente ejemplo para que encuentre los valores máximos y mínimos del array y los imprima por pantalla.

```
class Unidad4{
  public static void main(String[] args){
    int[] nums= new int[5];
    int min , max;
    nums[0]=7;
    nums[1]=10;
    nums[2]=3;
    nums[3]=34;
    nums[4]=13;
    .....................................
  }
}
```

Inicialización del array al mismo tiempo que se crea

Se puede inicializar un array al mismo tiempo que se crea. Por ejemplo

```
char[] vocales= {'a','e','i','o','u'};
```

con esta sintaxis, el compilador deduce que el array es de caracteres y que su tamaño es de 5, es decir, nos ahorramos escribir

new char[5]

además al mismo tiempo inicializamos el array ahorrándonos escribir

```
vocales[0]='a';
vocales[1]='b';
vocales[2]='c';
vocales[3]='d';
vocales[4]='e';
```

Ejercicio U4_B11_E3: modifica el ejemplo anterior de encontrar mínimo y máximo para que el array se inicialice cuando se cree.

Ejercicio U4_B11_E4: imprime la media del siguiente array

```
double[] notas = {8.5,7.0,6.0,9.2};
```

Ejercicio U4_B11_E5:
Utilizando los operadores lógicos, imprime las tablas de verdad de *a and b*, *a or b* , *a exor b y a nand b*. Las operaciones anteriores tienen su correspondiente operador java excepto *nand* que la implementamos manualmente como *not and*. Los valores de a y b deben de estar almacenados obligatoriamente para este ejercicio en arrays de tipo boolean con el siguiente aspecto.

a

false	false	true	true

b

false	true	false	true

Y la salida será (es suficiente con el and y el or):

```
L:\Programacion>java Unidad3
A       B       AorB    AandB   AexorB  AnandB
false   false   false   false   false   true
false   true    true    false   true    true
true    false   true    false   true    true
true    true    true    true    false   false

L:\Programacion>
```

Ejercicio U4_B11_E6: Acepta el reto ¿Cuántos días faltan? id 157
Podemos almacenar en un array los días que tiene cada mes por ejemplo:

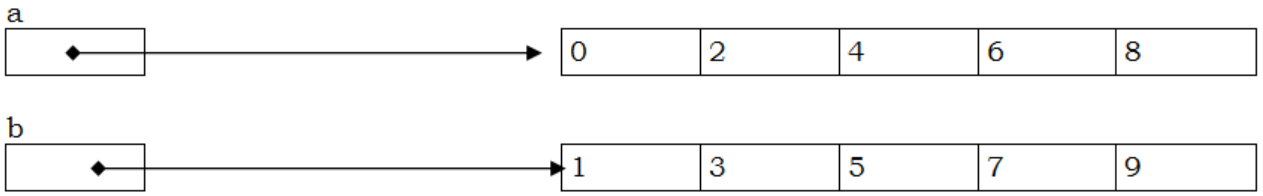
```
int[] diasMes={31,28,31,30,31,30,31,31,30,31,30,31};
```

https://www.aceptaelreto.com/problem/statement.php?id=157

Asignación entre referencias de Arrays
Los arrays son objetos, y accedemos a ellos a través de referencias. Examina este código

```
class Unidad4{
  public static void main(String[] args){
    int[]  a = {0,2,4,6,8};
    int[]  b = {1,3,5,7,9};
    System.out.println("Array referenciado por a: ");
    for(int i=0;i<5;i++)
      System.out.print(a[i] + "  ");
    System.out.println("\nArray referenciado por b: ");
    for(int i=0;i<5;i++)
      System.out.print(b[i] + "  ");
    System.out.println();
  }
}
```

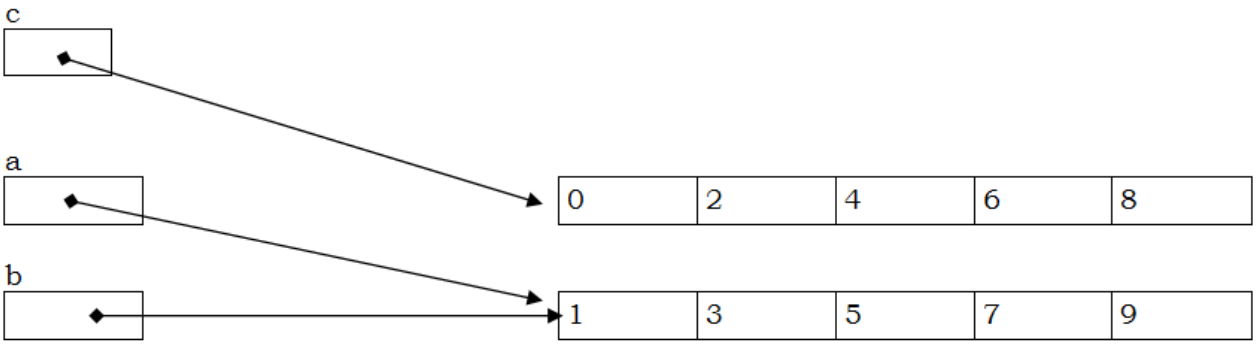
La situación en memoría es:



a y b son variables locales del main su valor estaŕa en la pila en la entrada correspondiente al método main. Los arrays son objetos que se almacenan en el heap

Ejercicio U4_B11_E7
Añade código al ejemplo anterior de forma que demuestre que si añado
int[] c;
c=a;
a=b;

la situación pasa a ser



Para demostrarlo imprime los Arrays a los que referencian a, b y c.
Conclusión: Las asignaciones entre referencias a arrays no provocan que se copien los arrays, lo que se copian son los valores de las referencias.

Copia de un array
Si queremos copiar un array, es decir, duplicarlo, visto el apartado anterior, está claro que no nos sirve una simple asignación de referencias.

Ejercicio U4_B11_E8
Continua el siguiente código para duplicar el array a en b (más exacto sería decir: para copiar el array referenciado por a en un array referenciado por b).

```
class Unidad4{
  public static void main(String[] args){
     int[] a= {1,2,3,4,5};
     int[] b= ....
     for(.....

  }
}
```

Uso del miembro length
En java rebasar los límites de un Array genera una excepción en tiempo de ejecución.

Ejemplo
Comprueba que se provoca una excepción al ejecutar el siguiente código.

```
class Unidad4{
  public static void main(String[] args){
    int[]  a ={1,2,3};
    for(int i=0;i<4;i++)
      System.out.print(a[i] + "  ");
  }
}
```

La comprobación que hace Java sobre los límites del array son una característica que mejora la seguridad de los programas, ya que evita entrar en posiciones de memoria más allá del array con resultados imprevisibles.
El atributo *length* almacena el número de elementos que el array puede contener. Es un atributo público y lo podemos utilizar en nuestros programas. Es de los pocos atributos públicos que hay en las clases del api java.

Ejemplo
Ejecuta este código

```
class Unidad4{
  public static void main(String[] args){
    int[]  a ={1,2,3};
    System.out.println("longitud del array a: " + a.length);
  }
}
```

Es cómodo utilizar el valor de este atributo para controlar el límite superior en un bucle

Ejemplo: Ejecuta el siguiente código

```
class Unidad4{
  public static void main(String[] args){
    int[]  a ={1,2,3};
    System.out.println(" longitud del array a: " + a.length);
    System.out.println(" El array es: " );
    for(int i = 0;i<a.length;i++)
      System.out.print(a[i] +  "  ");
  }
}
```

Ejercicio U4_B11_E9
Vuelve a escribir el siguiente ejemplo de copia de arrays utilizando la propiedad length.

```
class Unidad4{
  public static void main(String[] args){
```

```
    int[] a= {1,2,3,4,5};
    int[] b= new int[5]; //o también     int[] b= {0,0,0,0,0};
    for(int i=0;i<5;i++){
       b[i]=a[i];
       System.out.print("a[" + i +"]:"  + a[i]);
       System.out.println("      b[" + i +"]:"  + b[i]);
    }

  }
}
```

Escribe la solución de tres formas:

- todo en el main
- haciendo un método estático que haga la copia de los arrays. El método tendrá como parámetros el array origen y el array copia. Como los arrays son objetos, realmente no hay ninguna novedad ya que ya pasamos muchas veces una referencia por parámetro.
- haciendo un método estático que se le pasa por parámetro el array origen y devuelve en el return la copia

El array tiene un tamaño estático, pero como es un objeto, su tamaño se puede indicar dinámicamente(en tiempo de ejecución)

Hay que tener claro que el tamaño es estático, esto es, una vez que el array se crea no se puede disminuir ni aumentar su tamaño. En unidades posteriores veremos otros objetos similares al array que pueden variar su tamaño a lo largo de la ejecución del programa.
Lo que sí podemos es indicar el tamaño del array en tiempo de ejecución, por ejemplo, podemos especificar el tamaño del array a través de un valor que se introduce por teclado, y por tanto su tamaño puede variar en cada ejecución distinta del programa.
En el siguiente ejemplo, observamos como el tamaño del array se define en base a una variable *n* cuyo valor se introduce por teclado.

```
import java.util.Scanner;
class Unidad4{
  public static void main(String[]  args){
    Scanner teclado = new Scanner(System.in);
    System.out.println("cuantos números quieres guardar?");
    int n= teclado.nextInt();
    int[] numeros= new int[n];
    System.out.println("el tamaño del array es: "+ numeros.length);
    //una vez creado,  no hay forma de modificar el tamaño del array
    //numeros.length=9;
  }
}
```

La instrucción for y su sintaxis "especial arrays".

Realmente esta sintaxis es para objetos tipo "colección" no es exclusiva para Arrays. Un array es un tipo de colección. Con esta sintaxis, a la instrucción *for* también se le conoce como *"for-mejorado".* Hasta hace poco también se le llama for-each pero es mejor evitar ese nombre como entenderemos al estudiar streams.

```
  int[] numeros = {0,1,2,3,4};
  int suma=0;
  for(int i=0;i<numeros.length;i++)
     suma+=numeros[i];
```

Es equivalente a

```
int[] numeros = {0,1,2,3,4};
int suma=0;
for(int x: numeros)
   suma+=x;
```

El compilador detecta *int x:numeros* dentro de la cabecera del *for*. Los ":" delatan la nueva sintaxis e indican que queremos recorrer todo el array, una operación muy habitual. Con esta sintaxis java recorre automáticamente todo el array. En cada iteración del bucle, java lee el valor del array y se lo asigna a la variable x. Lógicamente x ha de ser del mismo tipo que los elementos del array, *int* en este caso.

Uso de esta sintaxis: siempre que se quiera recorrer el array para leerlo (sin modificarlo)
Observa que x es una variable local en la que se copia un valor del array, pero si modifico x no modifico el array

```
class Unidad4{
  public static void main(String[] args){
    int[] numeros= {1,2,3,4,5};
    for(int x:numeros){
       System.out.print(x);
       x=99;//esto modifica x no modifica el array
    }
     System.out.println("");
    for(int i=0;i<numeros.length;i++){
       int x=numeros[i];
       System.out.print(x);
       x=99;//no modificamos el array
    }
     System.out.println("");
    for(int x:numeros){
       System.out.print(x);
    }
  }
}
```

Ejercicio U4_B11_E10

Con las dos sintaxis anteriores, imprime por pantalla el siguiente array.

```
char[] vocales = {'a', 'e','i','o','u'};
```

Array vacío

Concepto similar a String vacío.
¿Qué es un array vacío?. Un array vacío es un array que existe pero tiene tamaño 0
¿Sí una referencia de array vale null quiere decir que referencia a un array vacío?. NO, que una referencia a un array valga null sólo quiere decir que esa referencia no referencia a ningún objeto array. El siguiente código genera un error en tiempo de ejecución.

```
class Unidad4{
   public static void main(String[] args) {
      String x1; //vale null
      String x2="";//String vacio

      int[] miArray=null;
      if(miArray.length==0)
         System.out.println("Array vacio");
   }
}
```

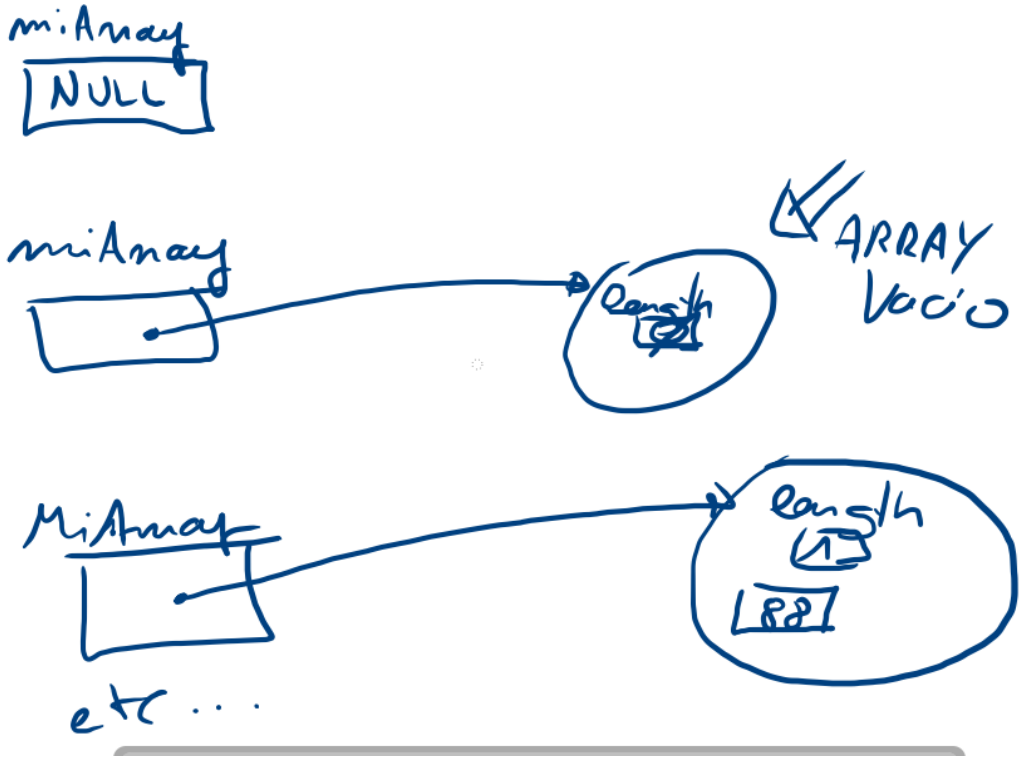
```
Exception in thread "main" java.lang.NullPointerException
        at Unidad4.main(Unidad4.java:4)
Java Result: 1
```

Siempre que obtenemos un NullPointerException es porque estamos intentando acceder a

un atributo o método con una referencia que vale null (sin ningún objeto asociado). En este caso miArray no tiene asignado ningún objeto array y por tanto es incongruente preguntar por su tamaño. Si quiero preguntar si un array tiene tamaño 0 podría haber escrito

```
class Unidad4{
    public static void main(String[] args) {
        int[] miArray= new int[0];
        if(miArray.length==0)
            System.out.println("Array vacio");
    }
}
```

Array vacio



observa que decimos array vacio o string vacio pero realmente estos objetos tiene cosas dentro como por ejemplo el código de los métodos. Lo de vacio se refiere a 0 posiciones almacenadas en el caso de los arrays y a 0 almacenados caracteres en el caso de los Strings.

Utilidades de arrays en el api para copiar y ordenar.

Copiar arrays de esta forma

```
class Unidad4{
    public static void main(String[] args) {
        char[] origen= { 'a', 'b', 'c', 'd', 'e' };
        char[] copia = new char[7];
        for(int i=0;i<origen.length;i++)
            copia[i]=origen[i];
        System.out.println(new String(copia));//forma fácil de imprimir array, pasándolo a String
    }
}
```

se puede evitar utilizando el método System.arraycopy(). Consulta el API y observa que el último parámetro se refiere al número de elementos que queremos copiar.

```
class Unidad4{
   public static void main(String[] args) {
      char[] origen= { 'a', 'b', 'c', 'd', 'e' };
      char[] copia = new char[7];
      System.arraycopy(origen, 0, copia, 0, origen.length);
      System.out.println(new String(copia));
   }
}
```

esta utilidad me permite además copiar fragmentos del original por ejemplo

```
class Unidad4{
   public static void main(String[] args) {
      char[] origen= { 'a', 'b', 'h', 'o', 'l','a','x' };
      char[] copia = new char[4];
      System.arraycopy(origen, 2, copia, 0, copia.length);
      System.out.println(new String(copia));
   }
}
```

```
hola
```

También hay un conjunto de utilidades para trabajar con Arrays en la clase Arrays

java.util

Class Arrays

java.lang.Object
    java.util.Arrays

que nos permite hacer copias, ordenamientos, búsqueda binarias,..

Para copiar también tenemos Arrays.copyof() que internamente usa System.arraycopy(). Arrays.copyof() es más sencillo pero más limitado. Nos quedamos con arraycopy

Ordenar con Arrays.sort()
La ordenación es un tema complejo que se estudiará más adelante. Por el momento simplemente saber que los arrays de tipos primitivos los podemos ordenar automáticamente con sort()

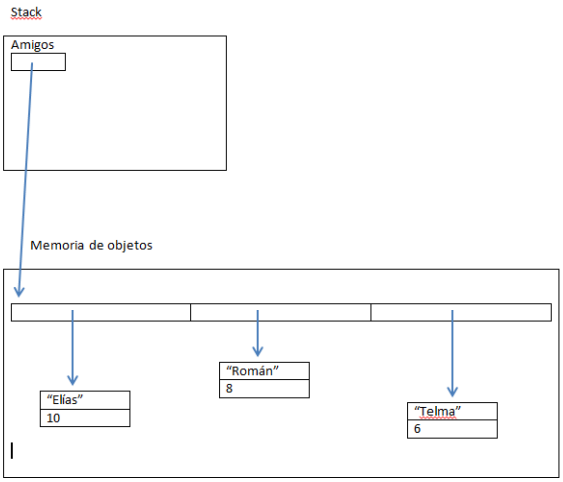
```
import java.util.Arrays;

class Unidad4{
  public static void main(String[] args) {
    int[] numeros={8,3,4,1,7};
    Arrays.sort(numeros);
    for(int i:numeros){
       System.out.print(i+" ");
    }
  }
}
```

Arrays de objetos
Si somos quisquillosos, hay que especificar que realmente con el término anterior nos queremos referir a *arrays de referencias*(a objetos). Es decir, "dentro" de una posición de un array de objetos realmente **no** se almacena un objeto, sino una referencia que nos permite acceder a un objeto, por lo tanto, el array almacena referencias y deberíamos decir array de referencias. En cualquier caso, el lenguaje se relaja y es habitual hablar de "arrays de objetos".

Ejemplo: Un array de objetos clase Persona, ilustrando varias formas de inicializarlo.

```
class Persona{
    String nombre;
    int edad;

    public Persona(String nombre, int edad) {
        this.nombre = nombre;
        this.edad = edad;
    }


}

class Unidad4{
  public static void main(String[] args) {
    //un array de amigos
    Persona[]  amigos = {new Persona("Elías",10), new Persona("Romàn",8), new Persona("Telma",6)};
    System.out.println("compruebo array amigos...");
    for(Persona p: amigos){
        System.out.println(p.nombre+ " "+p.edad);
    }

    //un array de enemigos
    Persona[] enemigos= new Persona[3];
    enemigos[0]=new Persona("chuli",7);
    enemigos[1]=new Persona("kachiu",9);
    enemigos[2]=new Persona("cesti",8);
    System.out.println("compruebo array enemigos...");
    for(Persona p: enemigos){
        System.out.println(p.nombre+ " "+p.edad);
    }

    //un array de locos
    System.out.println("compruebo array de locos...");
    Persona granLoco=new Persona("chifli",6);
    Persona[] locos={granLoco,new Persona("tarado total",9)} ;
    for(Persona p: locos){
        System.out.println(p.nombre+ " "+p.edad);
    }
  }
}
```

Es importante que entiendas bien lo que ocurre en memoria cuando escribimos
`Persona[]  amigos = {new Persona("Elías",10), new Persona("Romàn",8), new Persona("Telma",6)};`
Un dibujo, vale más que mil palabras:



Ejercicio U4_B11_E11
Crea un array de 5 amigos de la clase Persona. Pide datos por teclado. Utiliza un bucle para recoger los datos y muestra los datos del array por pantalla.

Ejercicio U4_B11_E12
Modifica el ejercicio anterior de forma que funcione para un número indeterminado de amigos. Al teclear un nombre de amigo "fin" se asume que ya no se quieren introducir más amigos. Como no sabes a priori el número de amigos que serán introducidos utiliza un array de tamaño 100 elementos. Tras introducir los amigos, el programa imprime el array.

Ejercicio U4_B11_E13
Repetimos la creación del array de amigos pero ahora no hay límite de amigos, simplemente, cada vez que se añade un amigo nuevo, creamos un nuevo array que contiene todos los amigos del array viejo y el nuevo amigo. Para copiar el array viejo en el nuevo puedes hacerlo "a mano" o utilizar System.arraycopy().

¿SON IMPORTANTES LOS ARRAYS?
Más adelante verás un montón de clases tipo colección de gran potencia(listas, mapas, etc.), que te pueden hacer pensar ¿para qué tanto array?. Pues a pesar de la existencia de esas sofisticadas clases, siempre nos va resultar necesario dominar los "humildes" arrays. Hay muchas razones que justifican su importancia, te voy a dar una muy inmediata y práctica: las clases del API utilizan muchísimo los arrays de forma que sus métodos frecuentemente tienen parámetros de tipo array, y como no te queda más remedio que manejar las clases del API, no te queda más remedio que utilizar arrays.

UN ARRAY DE CARACTERES NO ES UN STRING EN JAVA
Un array de caracteres en java podría ser
char[] saludo = {'h','o','l','a'}

El término *String* significa *"cadena"* (de caracteres). En muchos lenguajes, como en C, array de caracteres y String es lo mismo. En java el String se implementó como una clase propia para que fuera más fácil su manejo. Las literales String se representan como una tira de caracteres entre comillas dobles, por ejemplo, "hola" es un String y "hola\n" es otro String.
A menudo es necesario convertir un String en un array de char y viceversa. Hay múltiples formas de hacer conversiones entre ambos. Una forma muy común para convertir el array a String es usar una versión del contructor de String y para pasar de String a array usar el méto toCharArray()

```
class Unidad4 {
    public static void main(String[] args) {
        char[] saludo1 = {'h', 'o', 'l', 'a'};
        String saludo2 = new String(saludo1);
        char[] saludo3=saludo2.toCharArray();
        System.out.println(saludo1);
        System.out.println(saludo2);
        System.out.println(saludo3);
    }
}
```

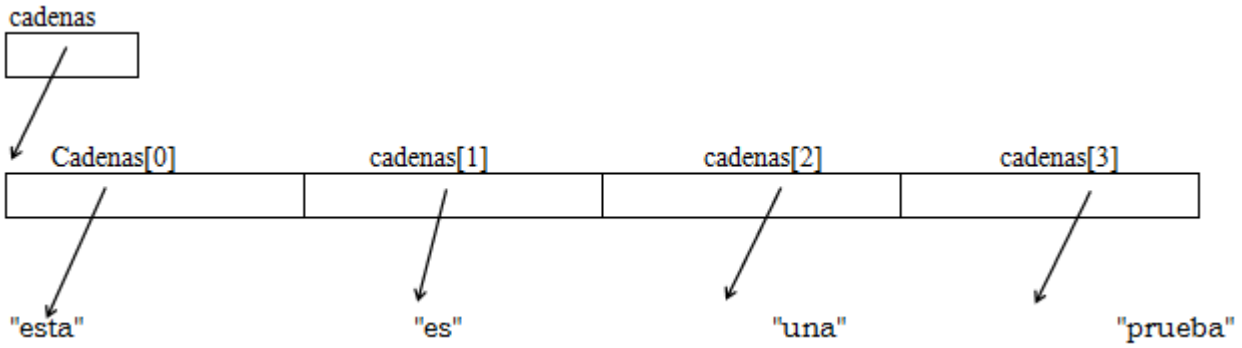
ARRAYS DE CADENAS

Puede haber arrays de cualquier tipo de dato primitivo u objeto: array de enteros, array de char, array de array de char, array de objetos Coche, ..., y también ¡array de cadenas! que por cierto son muy habituales.

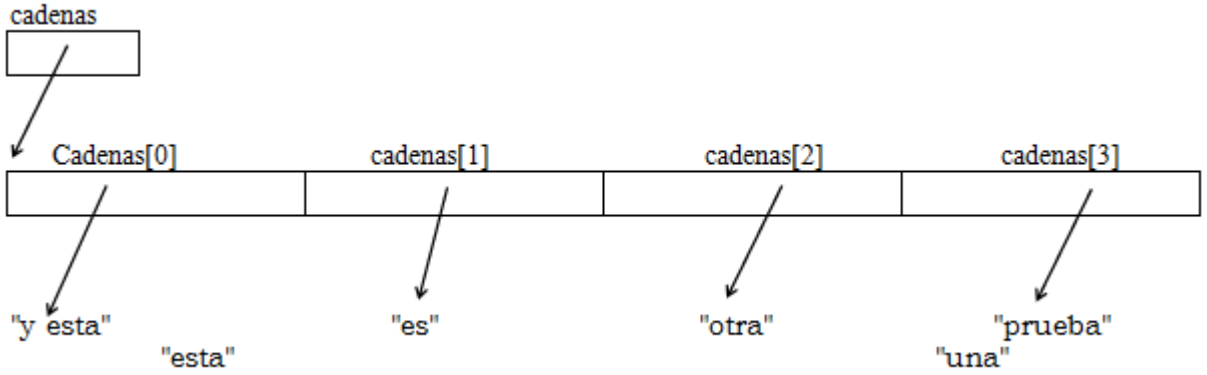
Ejemplo: Ejecuta el siguiente código

```
class Unidad4{
  public static void main(String[] args){
     String[] cadenas ={"Esta", "es", "una", "prueba." };
     System.out.println("Array  original: ");
     for(String s:cadenas)
        System.out.print(s + " ");
     cadenas[0]="y esta";
     cadenas[2]="otra";
    System.out.println("\nArray modificado: ");
    for(String s:cadenas)
       System.out.print(s + " ");

  }
}
```

Antes de la modificación la situación es:



Y después de la modificación:



los string "esta" y "una" quedarán perdidos en memoria, inaccesibles, sin ninguna variable referencia que los señale, y finalmente el recolector los eliminará en algún momento de memoria.

Ejercicio U4_B3_E14: Vuelve a escribir el ejemplo anterior con la sintaxis tradicional del for y sin utilizar la sintaxis de inicialización de arrays con {}.

SPLIT(). UN MÉTODO QUE DEVUELVE UN ARRAY DE CADENAS.

Es un método de Strings que se usa mucho. Split, se puede traducir por trocear. El método split() aplicado sobre un string devuelve dicho string troceado en substrings. Cada trozo es un substring de un array de strings. El criterio de troceo se le indica a split() como parámetro. Este criterio debe ser una expresión regular, pero como no sabemos lo que es una expresión regular, nos limitamos a utilizar como criterio de troceo un string que se limita a un carácter. Evitamos por razones que entenderemos al estudiar expresiones regulares el carácter "." (punto) como parámetro de split()

```
class Unidad4 {
    public static void main(String[] args) {
        String s="mi mama, me mima mucho";
        String[] split1=s.split(" ");
        String[] split2=s.split(",");
        String[] split3=s.split("m");
         System.out.println("split1 tiene "+split1.length+" trozos");
        for(String x: split1)
            System.out.println("\t"+x+" ");
         System.out.println("");
         System.out.println("split2 tiene "+split2.length+" trozos");
         for(String x: split2)
            System.out.println("\t"+x+" ");
         System.out.println("");
         System.out.println("split3 tiene "+split3.length+" trozos");

         for(String x: split3)
            System.out.println("\t"+x+" ");
         System.out.println("");
    }
}
```

ARGUMENTOS DE LÍNEA DE COMANDOS

En main(String[] args), estamos indicando que a main se le puede pasar como argumento un array de strings que se referencia con la variable *args*. Si no pasamos ningún parámetro simplemente el parámetro *args* apuntará a un array vacío. Recuerda que un array vacio es array con tamaño 0. Si args valiera null, obtendríamos una null pointer exception con args.length

Ejemplo:

```
class Unidad4{
  public static void main(String[] args){
     System.out.println("Hay " + args.length + " argumentos");
     for(int i=0;i<args.length;i++)
        System.out.println("args["+i+"]: " + args[i]);
  }
}
```

Ejecuta el código anterior desde la línea de comandos con los siguientes argumentos.

```
java Unidad4
java Unidad4 uno dos tres
java Unidad4 1 dos 3
java Unidad4 adios
java Unidad4 1,dos,3
```

Cada argumento es un string, y un argumento se distingue del anterior porque van separados por al menos un espacio en blanco

Ejemplo de ejecución

```
L:\Programacion>java Unidad3
Hay 0 argumentos

L:\Programacion>java Unidad3 uno dos tres
Hay 3 argumentos
args[0]: uno
args[1]: dos
args[2]: tres

L:\Programacion>java Unidad3 1 dos 3
Hay 3 argumentos
args[0]: 1
args[1]: dos
args[2]: 3

L:\Programacion>java Unidad3 1,dos,3
Hay 1 argumentos
args[0]: 1,dos,3

L:\Programacion>
```

Observa:

- cada argumento se interpreta como un String pero no se le ponen comillas
- que en la última ejecución *1,dos,3* se interpreta como un único argumento ya que los argumentos deben ir separados por espacios en blanco
- que el array args lo crea la máquina virtual java, en función de los argumentos teclados. El método main() lo recibe como parámetro, no lo crea.

Es típico ver que el array *args* se llama "args" ¡valga la redundancia!, pero podemos llamarlo como queramos, es decir, podemos escribir por ejemplo…

```
public static void main(String[] misArgumentos){ ……
```

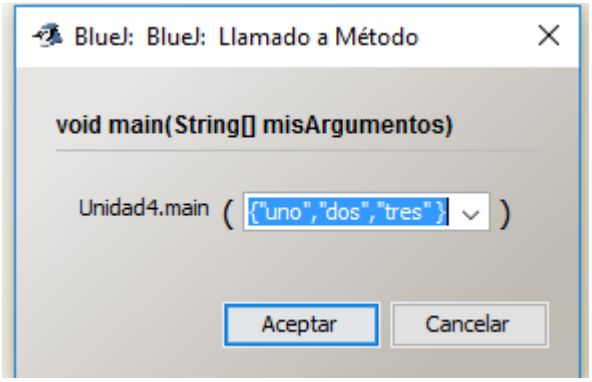
```
class Unidad4{
  public static void main(String[] misArgumentos){
     System.out.println("Hay " + misArgumentos.length + " argumentos");
     for(int i=0;i<misArgumentos.length;i++)
        System.out.println("misArgumentos["+i+"]: " + misArgumentos[i]);
  }
}
```

```
E:\programas\programasjavaconsola>java Unidad4 uno dos tres
Hay 3 argumentos
misArgumentos[0]: uno
misArgumentos[1]: dos
misArgumentos[2]: tres

E:\programas\programasjavaconsola>
```

En los ides, cada uno tendrá su forma de pasar parámetros al main, por ejemplo:

- netbeans
  menú->run->set project configuration ->costumize

- bluej se pasa con la sintaxis de incialización de arrays



Ejercicio U4_B3_E15:
Escribe un programa que le pasemos por consola una palabra y la rote hasta obtener de nuevo el original como en el siguiente ejemplo.

```
E:\programas\programasjavaconsola>java Unidad4 acueducto
acueducto
cueductoa
ueductoac
eductoacu
ductoacue
uctoacued
ctoacuedu
toacueduc
oacueduct
acueducto
```

Otra sintaxis para inicializar arrays.
observa cómo se inicializa *a2*. En este ejemplo no es necesaria y es preferible como se hace a1 ya que es más simple, pero en ciertos contextos, es necesaria la segunda forma.

```
class Unidad4{
   public static void main(String[] args) {
      int[] a1={1,2,3,4};
      int[] a2= new int[]{1,2,3,4};
   }
}
```

Concretamente la segunda forma es necesaria cuando queremos crear e inicializar un array

sin asociarlo a una variable referencia, por ejemplo en un return

```
class Unidad4{
   static int[] crearArray(){
      //return {1,2,3,4};   ERROR
      return new int[]{1,2,3,4}; //OK
   }
   public static void main(String[] args) {
      int[] a1=crearArray();

   }
}
```

o por ejemplo en un constructor o método con un parámetro array

```
class Cliente{
   String nombre;
   String[] telefonos;
   Cliente(String nombre,String[] telefonos){
      this.nombre=nombre;
      this.telefonos=telefonos;
   }
}
class Unidad4{
   public static void main(String[] args) {
      //Cliente c1= new Cliente("yo", {"981-111","699-111"}); ERROR
      Cliente c2= new Cliente("yo", new String[] {"981-111","699-111"});
      //otra forma con más pasos que no requiere sintáxis anterior
      String[] telefonos={"981-111","699-111"};
      Cliente c3= new Cliente("yo",telefonos);
   }
}
```