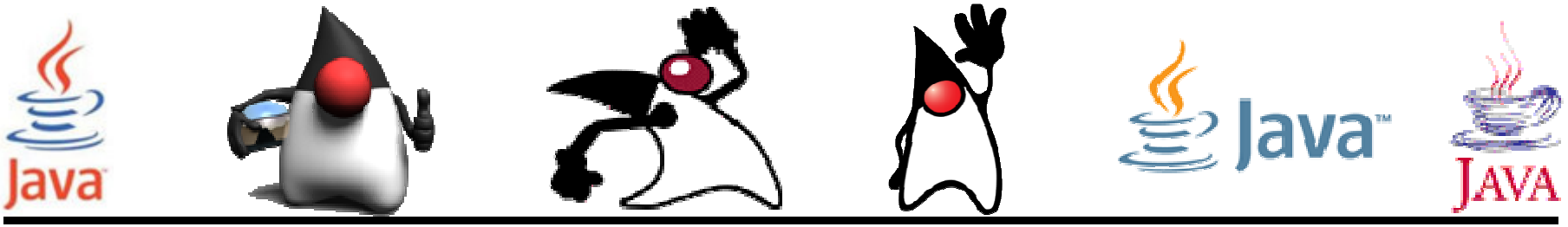


Arrays y ArrayList

Pedro Corcuera

Dpto. Matemática Aplicada y
Ciencias de la Computación

Universidad de Cantabria

corcuerp@unican.es

Objetivos

- Familiarizarse con el uso de arrays y array lists para coleccionar valores.
- Usar el ciclo for para el recorrido de arrays y array lists.
- Aprender algoritmos comunes para el procesado de arrays y array lists.
- Usar arrays multidimensionales.

Índice

- Arrays
- Ciclo for mejorado
- Algoritmos con arrays
- Uso de arrays con métodos
- Arrays multidimensionales
- Array Lists

¿Qué es un array?

- Es usual en los programas la necesidad de almacenar una lista de valores para después procesarlos.
- Una posibilidad es asociar a cada valor una variable, pero esto sería ineficiente y engorroso.
- Un **array** es una variable que almacena una lista de valores del mismo tipo.
- El array se almacena en posiciones continuas de memoria y el acceso a los elementos se realiza mediante índices.

Declaración de Arrays

- Para declarar un array se requiere el tipo de dato de los elementos a almacenar y un nombre para el array.
- Sintaxis:

```
double[] data; // declara var. array data
```

o

```
double data[];
```

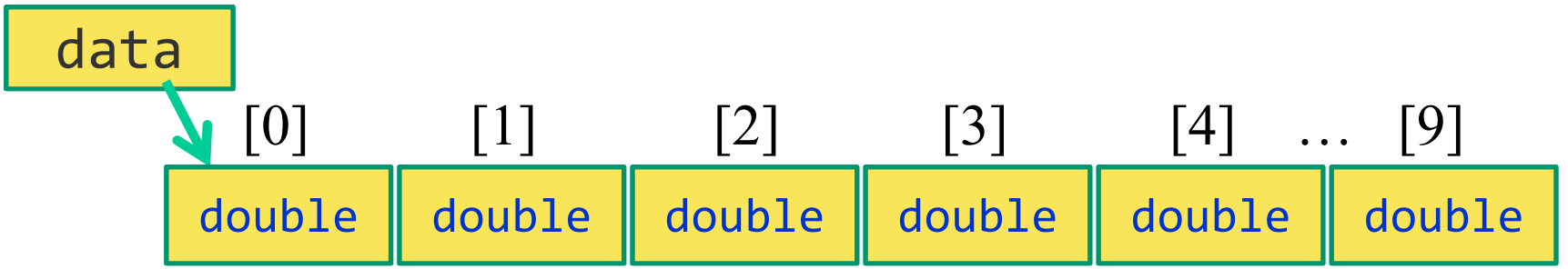
Tipo	Corchetes	Nombre Array	punto y coma
`double`	`[ ]`	`data`	`;`

Creación de un array (instancia)

- Después de declarar un array es necesario reservar memoria para todos los elementos.
- Se especifica el número de elementos del array a través de un método constructor (new).

Nota: No se puede cambiar el tamaño después de crear el array

Nombre Array		Pal.reservada	Tipo	Tamaño	punto y coma
`data`	=	`new`	`double`	`[10]`	`;`



Declaración y Creación de un array

- Se puede declarar y crear un array al mismo tiempo:

Tipo	Corchetes	Nombre Array		Constructor	Tipo	Tamaño	pyc
`double`	`[]`	`data`	`=`	`new`	`double`	`[10]`	`;`

Declaración y Creación de un array

- Se puede declarar y asignar el valor inicial de todos los elementos:

Tipo	Corchetes	Nombre Array		Lista del contenido	pyc
`int`	`[ ]`	`primos`	=	`{ 2, 3, 5, 7}`	`;`

- Se declara:
  - Nombre del array : `primos`
  - Los elementos del array son del tipo: `int`
  - Reserva espacio para cuatro elementos
    - El compilador los cuenta
  - Asigna valores iniciales a 2, 3, 5 y 7. Notar las llaves

Ejemplos de declaración

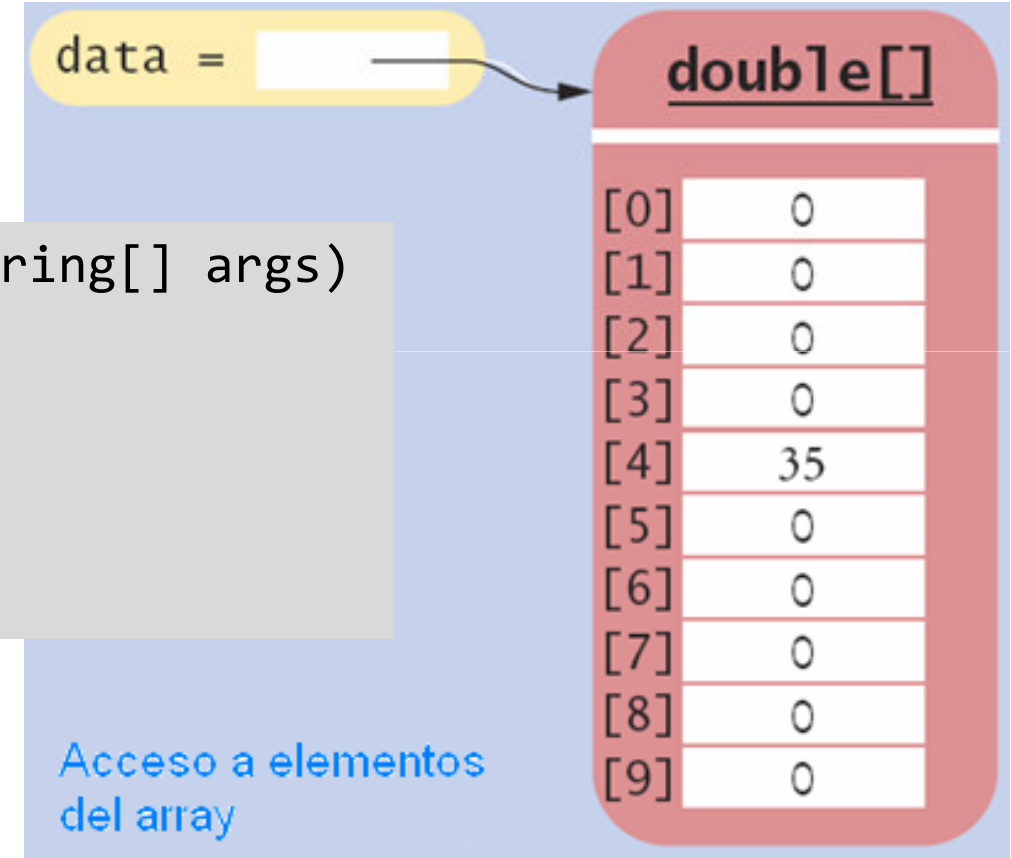
```
//creacion y asignacion de un array de 4 valores
//booleanos
  boolean resultados[] = {true,false,true,false};
//creacion y asignacion de un array de 4 valores
//double
  double[] notas = {100, 90, 80, 75};
//creacion y asignacion de un array de 7 cadenas
//de caracteres
  String dias[] = {"Lun", "Mar", "Mie", "Jue", "Vie",
  "Sab", "Dom"};
```

Acceso a elementos de un array

- Cada elemento del array está numerado mediante un *índice*, de tipo entero, que empieza en 0 y progresa secuencialmente hasta tamaño_array − 1.
  - Cuando se declaran y construyen arrays de datos numéricos todos los elementos se inicializan a 0.
  - Para tipos de datos referencia como los Strings se deben inicializar explícitamente.
- Para acceder a un elemento del array se usa:
  `data[i]`

Acceso a elementos de un array

```
public static void main(String[] args)
{
  double data[];
  data = new double[10];
  data[4] = 35;
}
```



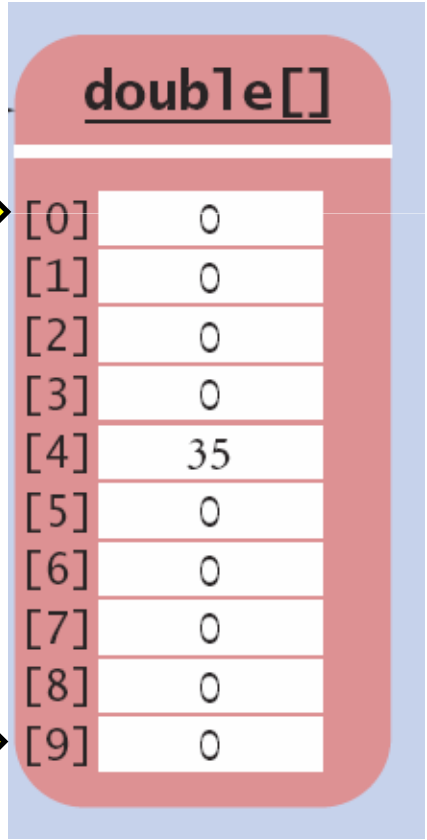
Números de índice de array

- El índice de un array empieza en 0.
- Un array de n elementos tiene como rango de índice 0 a n − 1

El primer elemento está en el índice 0

```
public static void main(String[] args)
{
  double data[];
  data = new double[10];
}
```

El último elemento está en el índice 9



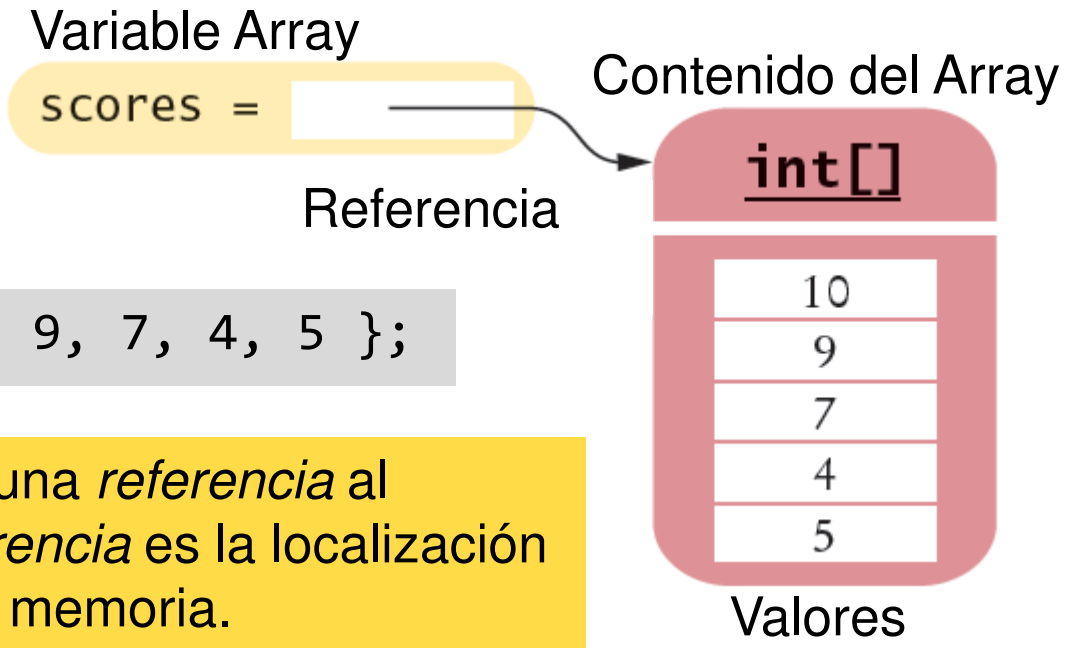
Longitud del array

- Un array sabe cuántos elementos puede almacenar con `data.length` donde `data` es el nombre del array.
- Se puede usar para comprobar el rango y prevenir errores de límites.

```
public static void main(String[] args)
{
  int i = 10, value = 34;
  double data[] = new double[10];
  if (0 <= i && i < data.length) {  // valor es 10
    data[i] = value;
  }
}
```

Referencias a arrays

- Diferencia entre:
  - Variable array: El nombre del array (manejador).
  - Contenido del array: Memoria donde se almacenan los valores



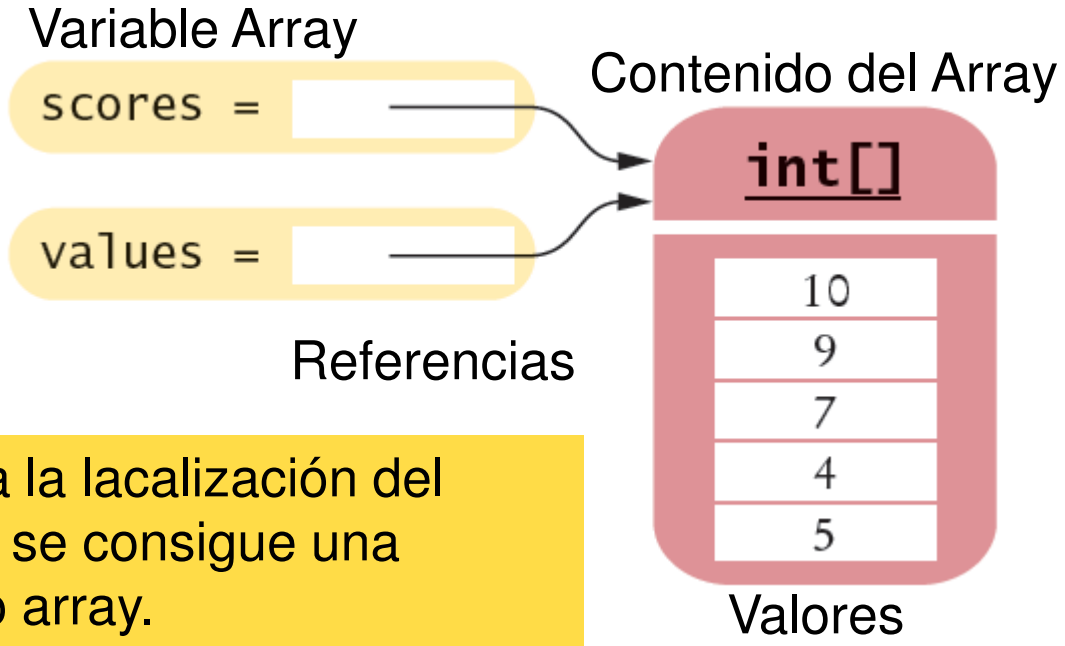

```
int scores[] = { 10, 9, 7, 4, 5 };
```

Una variable array contiene una *referencia* al contenido del array. La *referencia* es la localización del contenido del array en la memoria.

Alias de arrays

- Se puede hacer que una referencia de array se refiera al mismo contenido de otro.

```
int scores[] = { 10, 9, 7, 4, 5 };
int values[] = scores;  // Copia de la ref. del array
```



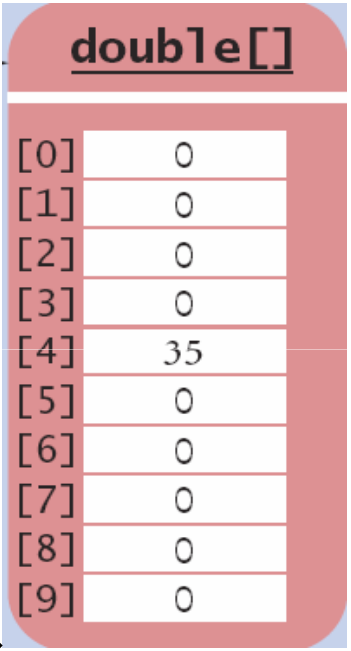
Una variable array especifica la lacalización del array. Al copiar la referencia se consigue una segunda referencia al mismo array.

Errores comunes con arrays

- Errores de límites del array.
  - Acceder a un elemento inexistente.
  - Se produce en error en tiempo de ejecución.

```
public class OutOfBounds
{
  public static void main(String[] args)
  {
    double data[] = new double[10];
    data[10] = 100;
  }
}
```

No hay elemento 10. ERROR!




```
java.lang.ArrayIndexOutOfBoundsException: 10
        at OutOfBounds.main(OutOfBounds.java:7)
```

Errores comunes con arrays

- Arrays sin crear
  - Es frecuente olvidarse de asignar memoria para el contenido del array.
  - Se produce en error en tiempo de compilación.

```
double data[];
...
data[0] = 29.95; // Error—datos sin inicializar
```

```
Error: D:\Java\Unitialized.java:7:
variable data might not have been initialized
```

```
double data[];
data = new double[10];
data[0] = 29.95; // Sin error
```

Recomendaciones de codificación

- Declarar las dimensiones de los arrays usando constantes para facilitar las modificaciones.

```
final int ARRAY_SIZE = 1000; //declara una constante
...
int edades[] = new int[ARRAY_SIZE];
```

- Cuando se usan for para el recorrido de una array usar `array.length` en la condición del for.

```
int edades[] = new int[100];
for (int i=0; i < edades.length; i++)
{
   ...
}
```

Ciclo for mejorado – for each

- Hay un ciclo for, llamado for each, que permite acceder a cada elemento del array secuencialmente.
- No permite modificar un elemento del array.

```
double[] data = . . .;
double sum = 0;
for (double element : data)
{
   sum = sum + element;
}
```

Esta variable es asignada a cada elemento del array en cada iteración del ciclo. Está definida sólo dentro del ciclo

Arrays multidimensionales

- Un array multidimensional es tratado como un array de arrays.
- Los arrays multidimensionales se declaran colocando un número de corchetes igual a la dimensión del array antes/después del nombre del array.

```
//array de doubles de 512x128 elementos
  double twoD[][] = new double[512][128] ;
//array de caracteres de 8x16x24 elementos
  char[][][] threeD = new char[8][16][24];
//declaracion e inicializacion de una matriz
  double[][] m1 = {{1,2,3},{4,5,6}};
```

Arrays multidimensionales - Declaración

- Declaración e instanciación.

```
const int PAISES = 7;
const int MEDALLAS = 3;
int[][] cuenta = new int[PAISES][MEDALLAS];
```

- Declaración e inicialización.

```
const int PAISES = 7; const int MEDALLAS = 3;
int[][] cuenta = {{ 0, 0, 1 },
                  { 0, 1, 1 },
                  { 1, 0, 0 },
                  { 3, 0, 1 },
                  { 0, 1, 0 },
                  { 0, 0, 1 },
                  { 0, 2, 0 }
                 };
```

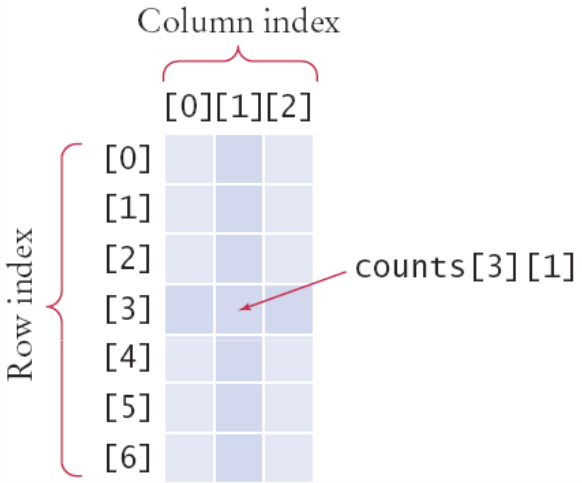
Gold	Silver	Bronze
0	0	1
0	1	1
1	0	0
3	0	1
0	1	0
0	0	1
0	2	0

Arrays multidimensionales - Acceso

- El acceso a un elemento de un array md es igual que en un array unidimensional.
- Caso bidimensional

Fila Columna

```
int valor = cuenta[3][1];
```




```
for (int i = 0; i < PAISES; i++) {// Proceso fila ith
  for (int j = 0; j < MEDALLAS; j++) {
    // Procesa la jth columna en la fila ith
    System.out.printf("%8d", cuenta[i][j]);
  }
  System.out.println(); // Cambio línea al final de la fila
}
```

Ejemplos de array bidimensional

```
public class Medallas {
   public static void main(String[] args) {
      final int PAISES = 7;
      final int MEDALLAS = 3;
      String[] paises = {"Canada","China",
  "Japon","Rusia","Espana","Ucrania","Estados
  Unidos" };
      int[][] cuentas = { { 0, 0, 1 },{ 0, 1, 1
  }, { 1, 0, 0 }, { 3, 0, 1 }, { 0, 1, 0 }, { 0,
  0, 1 }, { 0, 2, 0 }};
  System.out.println("          Pais       Oro
  Plata  Bronce   Total");
```

Ejemplos de array bidimensional

```
        for (int i = 0; i < PAISES; i++) {
            System.out.printf("%15s", paises[i]);
            int total = 0;
            for (int j = 0; j < MEDALLAS; j++) {
                System.out.printf("%8d",cuentas[i][j]);
                total = total + cuentas[i][j];
            }
            System.out.printf("%8d\n", total);
        }
    }
}
```

Algoritmos comunes

- Relleno de un array

```
int[] data = new int[11];
for (int i = 0; i < data.length; i++)
{
  data[i] = i * i;
}
```

- Suma y promedio

```
double total = 0, promedio = 0;
for (double elemento : data)
{
  total = total + elemento;
}
if (data.length > 0) { promedio = total / data.length; }
```

Algoritmos comunes

- Máximo y mínimo

```
double maximo = data[0];
for (int i = 1; i < data.length; i++) {
  if (data[i] > maximo) {
    maximo = data[i];
  }
}
```

- Uso de for each

```
double maximo = data[0];
for (double element : data)
{
  if (element > maximo)
    maximo = element;
}
```

```
double minimo = data[0];
for (double element : data)
{
  if (element < minimo)
    minimo = element;
}
```

Algoritmos comunes

- Separador de elementos

```
for (int i = 0; i < data.length; i++) {
  if (i > 0) {
    System.out.print(" | ");
  }
  System.out.print(data[i]);
}
```

- Método manejar arrays: `Arrays.toString()` útil para depuración

```
import java.util.*;
System.out.println(Arrays.toString(data));
```

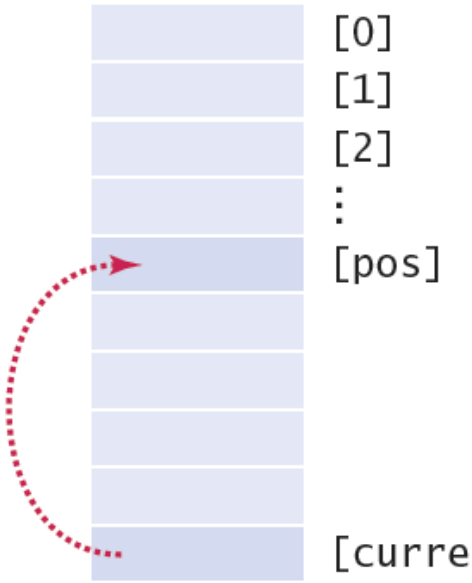
Algoritmos comunes

- Búsqueda lineal

```
int valorBuscado = 100;
int pos = 0;
boolean found = false;
while (pos < data.length && !found)
{
  if (data[pos] == valorBuscado) { found = true; }
  else { pos++; }
  if (found)
  {
    System.out.println("Hallado en la posicion: " + pos);
  }
    else { System.out.println("No encontrado");
}
```

Algoritmos comunes

- Eliminación de un elemento
  - Requiere el seguimiento del ‘ currentSize’ (número de elementos válidos).
  - La solución depende si se tiene que mantener el orden

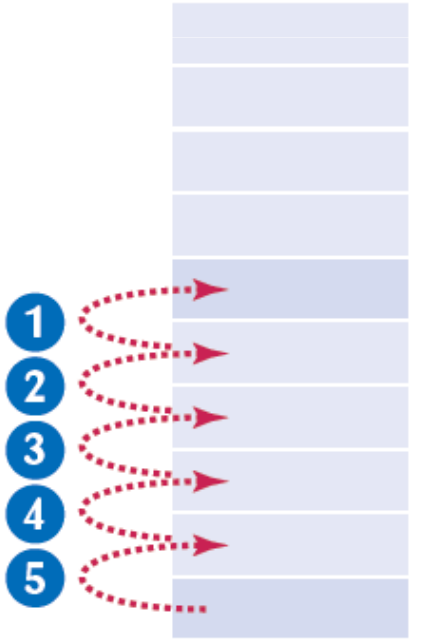



```
data[pos] = data[currentSize – 1];
currentSize--;
```

Algoritmos comunes

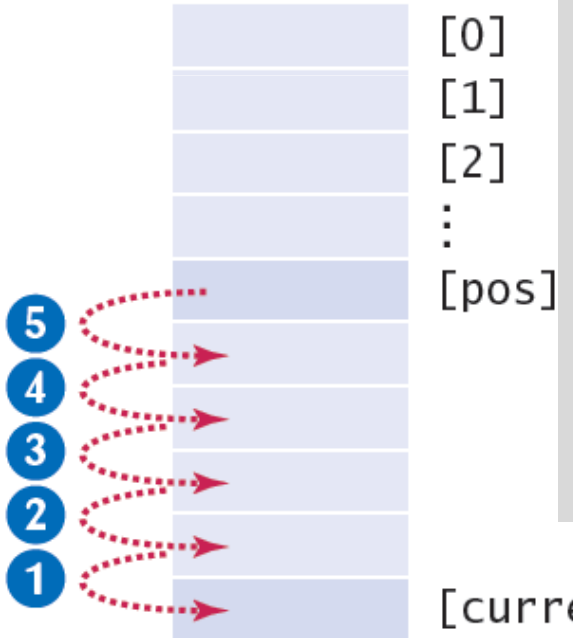
- Eliminación de un elemento

```
for (int i = pos; i < currentSize - 1; i++)
{
  data[i] = data[i + 1];
}
currentSize--;
```



Algoritmos comunes

- Inserción de un elemento
  - Si se quiere conservar el orden, sino añadir al final.

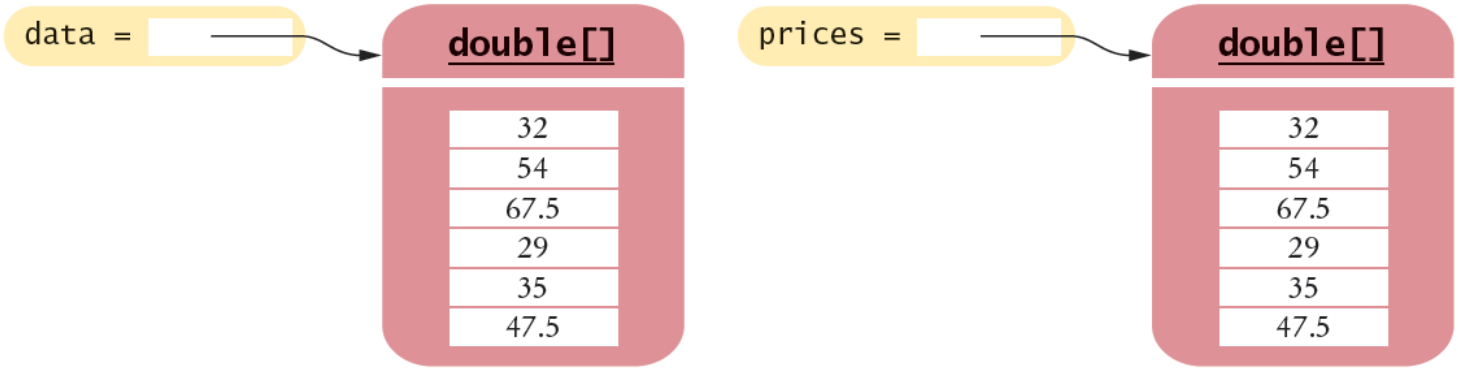



```
if (currentSize < data.length)
{
  for (int i = currentSize; i > pos; i--)
  {
    data[i] = data[i - 1];
  }
  data[pos] = newElement;
  currentSize++;
}
```

Algoritmos comunes

- Copia del contenido de un array en otro
  - Uso del nuevo método (Java 6) `Arrays.copyOf`

```
import java.util.Arrays;
. . .
double[] data = new double[6];
. . . // Llenado del array
double[] prices = data;   // Solo la referencia
// copyOf crea la copia, devuelve una referencia
double[] prices = Arrays.copyOf(data, data.length);
```



Algoritmos comunes

- Aumento del tamaño de un array
  - Copiar los elementos del array a uno mayor.
  - Cambiar la referencia del array original al más grande.

```
import java.util.Arrays; // doblar el tamaño original
. . .
double[] newData = Arrays.copyOf(data, 2 * data.length);
data = newData;
```

El segundo parámetro de `Arrays.copyOf` es la longitud del nuevo array

Algoritmos comunes

- Lectura de valores del array
  - Si se conoce el número de valores

```
double[] inputs = new double[NUMBER_OF_INPUTS];
for (i = 0; i < values.length; i++)
{
  inputs[i] = in.nextDouble();
}
```

  - Si no se conoce el número de valores (estimar un máximo)

```
double[] inputs = new double[MAX_INPUTS];
int currentSize = 0;
while (in.hasNextDouble() && currentSize < inputs.length){
  inputs[currentSize] = in.nextDouble();
  currentSize++;
}
```

Algoritmos comunes - Ordenación

- Ordenación o clasificación es el proceso de reordenar un conjunto de objetos en un orden específico.
- El propósito de la ordenación es facilitar la búsqueda de elementos en el conjunto ordenado.
- Existen muchos algoritmos de ordenación, siendo la diferencia entre ellos la eficiencia en tiempo de ejecución.
- Los métodos de ordenación se pueden clasificar en dos categorías: ordenación de ficheros o externa y *ordenación de arrays* o interna.

Algoritmos comunes - Ordenación

- Formalmente el problema del ordenamiento se expresa como:
  - Dados los elementos: $a_1, a_2, \ldots, a_n$
  - Ordenar consiste en permutar esos elementos en un orden: $a_{k_1}, a_{k_2}, \ldots, a_{k_n}$ tal que dada una función de ordenamiento f: $f(a_{k_1}) \leq f(a_{k_2}) \leq \ldots \leq f(a_{k_n})$
- Normalmente, la función de ordenamiento se guarda como un componente explícito (campo) de cada item (elemento). Ese campo se llama la *llave del item*.
- Un método de ordenamiento es *estable* si el orden relativo de elementos con igual llave permanece inalterado por el proceso de ordenamiento.

Algoritmos comunes - Ordenación

- Los métodos de ordenación buscan un uso eficiente de la memoria por lo que las permutaciones de elementos se hará *in situ* (uso del array original).
- Existen varios métodos de ordenación: burbuja, agitación, selección, inserción, quicksort, etc.

http://personales.unican.es/corcuerp/ProgComp/Ordena/AlgoritmosOrdenamiento.html

http://www.cs.ubc.ca/~harrison/Java/sorting-demo.html

- La API de Java API ofrece un método de ordenación eficiente (ascendente por defecto):

```
Arrays.sort(data); // Ordenacion de todo el array
Arrays.sort(data, 0, currentSize);   // parcial
```

Análisis de Algoritmos: Complejidad

- Para comparar algoritmos se pueden estudiar desde dos puntos de vista:
  - el tiempo que consume un algoritmo para resolver un problema (complejidad temporal) ⟵ más interés
  - la memoria que necesita el algoritmo (complejidad espacial).
- Para analizar la complejidad se cuentan los pasos del algoritmo en función del tamaño de los datos y se expresa en unidades de tiempo utilizando la notación asíntotica “O- Grande” (complejidad en el peor caso).

Análisis de Algoritmos: Complejidad

Problema: Buscar el mayor valor en una lista de números desordenados (array)

```
Algoritmo: (n = número de elementos)
1   max = s1
2   i = 2
3   while i <= n
4     if si > max then
5               max = si
6       i = i + 1
7   endwhile
```

Análisis de Algoritmos: Complejidad

Número de operaciones realizadas (unid):

Línea	Operaciones	Tiempo
1	indexado y asignación	2
2	asignación	1
3	comparación	1
4,5,6	2 indexado, comparación, 2 asignación, suma	6

Tiempo total:

$t(n) = 2 + 1 + (n - 1) + 6 \cdot (n - 1)$

$= 3 + 7 \cdot (n - 1) = 7n - 4$

- Sean f(n) y g(n) funciones no negativas, f(n) es O(g(n)) si hay un valor $c > 0$ y $n_0 \geq 1$ tal que $f(n) \leq cg(n)$ para $n \geq n_0$
- Se dice que f(n) es de orden g(n)
- Ej: $7n - 4$ es O(n) si $c=7$ y $n_0 = 1$

Método de Ordenación: burbuja

- Es un método caracterizado por la *comparación* e *intercambio* de pares de elementos hasta que todos los elementos estén ordenados.
- En cada iteración se coloca el elemento más pequeño (orden ascendente) en su lugar correcto, cambiándose además la posición de los demás elementos del array.
- La complejidad del algoritmo es $O(n^2)$.

Método de Ordenación: burbuja

Original	1ª iter	2ª iter	3ª iter	4ª iter	5ª iter	6ª iter	7ª iter
44							
55							
12							
42							
94							
18							
06 →	06						
67 →	67						

67 < 06 no hay intercambio

Método de Ordenación: burbuja

Original	1ª iter	2ª iter	3ª iter	4ª iter	5ª iter	6ª iter	7ª iter
44							
55							
12							
42							
94							
18	06						
06	18						
67	67						

06 < 18 hay intercambio

Método de Ordenación: burbuja

Original	1ª iter	2ª iter	3ª iter	4ª iter	5ª iter	6ª iter	7ª iter
44							
55							
12							
42							
94	06						
18	94						
06	18						
67	67						

06 < 94 hay intercambio

Método de Ordenación: burbuja

Original	1ª iter	2ª iter	3ª iter	4ª iter	5ª iter	6ª iter	7ª iter
44							
55							
12							
42	06						
94	42						
18	94						
06	18						
67	67						

06 < 42 hay intercambio

Método de Ordenación: burbuja

Original	1ª iter	2ª iter	3ª iter	4ª iter	5ª iter	6ª iter	7ª iter
44							
55							
12	06						
42	12						
94	42						
18	94						
06	18						
67	67						

06 < 12 hay intercambio

Método de Ordenación: burbuja

Original	1ª iter	2ª iter	3ª iter	4ª iter	5ª iter	6ª iter	7ª iter
44							
55	06						
12	55						
42	12						
94	42						
18	94						
06	18						
67	67						

06 < 55 hay intercambio

Método de Ordenación: burbuja

Original	1ª iter	2ª iter	3ª iter	4ª iter	5ª iter	6ª iter	7ª iter
44	06						
55	44						
12	55						
42	12						
94	42						
18	94						
06	18						
67	67						

06 < 44 hay intercambio

Método de Ordenación: burbuja

Original	1ª iter	2ª iter	3ª iter	4ª iter	5ª iter	6ª iter	7ª iter
44	06	06					
55	44	12					
12	55	44					
42	12	55					
94	42	18					
18	94	42					
06	18	94					
67	67	67					

Método de Ordenación: burbuja

Original	1ª iter	2ª iter	3ª iter	4ª iter	5ª iter	6ª iter	7ª iter
44	06	06	06				
55	44	12	12				
12	55	44	18				
42	12	55	44				
94	42	18	55				
18	94	42	42				
06	18	94	67				
67	67	67	94				

Método de Ordenación: burbuja

Original	1ª iter	2ª iter	3ª iter	4ª iter	5ª iter	6ª iter	7ª iter
44	06	06	06	06	06	06	06
55	44	12	12	12	12	12	12
12	55	44	18	18	18	18	18
42	12	55	44	42	42	42	42
94	42	18	55	44	44	44	44
18	94	42	42	55	55	55	55
06	18	94	67	67	67	67	67
67	67	67	94	94	94	94	94

Método de Ordenación: burbuja

```
public class OrdBurbuja {
  public static void main(String args[]){
    double data[]={321,123,213,234,1,4,5,6}; //Array a ordenar

    for(int i = 0; i < data.length; i++)
      for(int j = data.length-2; j >= i; j--)
        if (data[j] > data[j+1]) { /* orden ascendente */
          double tmp = data[j];
          data[j] = data[j+1];
          data[j+1] = tmp;
        }

    for (int i = 0; i < data.length; i++) { //Imprime array orden
      System.out.println(data[i]);
    }
  }
}
```

Núcleo del algoritmo

Método de Ordenación: inserción

- Método usado para ordenar una mano de naipes.
- Los elementos están divididos conceptualmente en una secuencia destino y una secuencia fuente.
- En cada paso, comenzando con i=2 e incrementando i en uno, el elemento i-ésimo de la secuencia fuente se toma y se transfiere a la secuencia destino insertándolo en el lugar adecuado.
- Este algoritmo puede mejorarse fácilmente si vemos que la secuencia destino $a_1, a_2, \ldots, a_{i-1}$ está ordenada, por lo que usamos una búsqueda binaria para determinar el punto de inserción.
- La complejidad del algoritmo es $O(n^2)$. Es estable.

Método de Ordenación: inserción

Or	44	55	12	42	94	18	06	67
1	44	55	12	42	94	18	06	67
2								
3								
4								
5								
6								
7								

Método de Ordenación: inserción

Or	44	55	12	42	94	18	06	67
1	44	55	12	42	94	18	06	67
2	12	44	55	42	94	18	06	67
3								
4								
5								
6								
7								

Método de Ordenación: inserción

Or	44	55	12	42	94	18	06	67
1	44	55	12	42	94	18	06	67
2	12	44	55	42	94	18	06	67
3	12	42	44	55	94	18	06	67
4								
5								
6								
7								

Método de Ordenación: inserción

Or	44	55	12	42	94	18	06	67
1	44	55	12	42	94	18	06	67
2	12	44	55	42	94	18	06	67
3	12	42	44	55	94	18	06	67
4	12	42	44	55	94	18	06	67
5	12	18	42	44	55	94	06	67
6	06	12	18	42	44	55	94	67
7	06	12	18	42	44	55	67	94

Método de Ordenación: inserción

```
public class OrdInsercion {
  public static void main(String args[]) {
     double data[]={321,123,213,234,1,4,5,6}; //Array a ordenar

     for(int i = 1; i < data.length; i++) {
       int j = i-1;
       double tmp = data[i]; // Elemento a insertar
       while (j >= 0 && tmp < data[j]) {
         data[j+1] = data[j]; j = j-1;
       }
       data[j+1] = tmp;
     }
     for (int i = 0; i < data.length; i++) { //Imprime array orden
       System.out.println(data[i]); }
  }
}
```

Método de Ordenación: selección

- En éste método, en el i-ésimo paso seleccionamos el elemento con la llave de menor valor, entre a[i],…, a[n] y lo intercambiamos con a[i].
- Como resultado, después de i pasadas, el i-ésimo elemento menor ocupará a[1],…, a[i] en el lugar ordenado.
- La complejidad del algoritmo es $O(n^2)$.

Método de Ordenación: selección

Or	44	55	12	42	94	18	06	67
1	06	55	12	42	94	18	44	67
2								
3								
4								
5								
6								
7								

Método de Ordenación: selección

Or	44	55	12	42	94	18	06	67
1	06	55	12	42	94	18	44	67
2	06	12	55	42	94	18	44	67
3								
4								
5								
6								
7								

Método de Ordenación: selección

Or	44	55	12	42	94	18	06	67
1	06	55	12	42	94	18	44	67
2	06	12	55	42	94	18	44	67
3	06	12	18	42	94	55	44	67
4								
5								
6								
7								

Método de Ordenación: selección

Or	44	55	12	42	94	18	06	67
1	06	55	12	42	94	18	44	67
2	06	12	55	42	94	18	44	67
3	06	12	18	42	94	55	44	67
4	06	12	18	42	94	55	44	67
5	06	12	18	42	44	55	94	67
6	06	12	18	42	44	55	94	67
7	06	12	18	42	44	55	67	94

Método de Ordenación: selección

```
public class OrdSeleccion {
  public static void main(String args[]) {
     double data[]={321,123,213,234,1,4,5,6}; //Array a ordenar

     for (int sinord = 0; sinord < data.length - 1; sinord ++) {
       int minPos = sinord ; // Halla la posición del mínimo
       for (int i = sinord + 1; i < data.length; i++) {
         if (data[i] < data[minPos]) { minPos = i; }
       }
       if (minPos != sinord ) { double temp = data[minPos];
        data[minPos] = data[sinord ]; data[sinord ] = temp;
       } // Intercambio
     }
     for (int i = 0; i < data.length; i++) { //Imprime array orden
       System.out.println(data[i]); }
  } }
```

Método de Ordenación: Quicksort

- Se basa en el hecho que los intercambios deben ser realizados preferentemente sobre distancias grandes.
- El algoritmo (técnica de dividir y vencer) simplificado es:
  - Seleccionar un elemento del array (elemento pivote, p.e. el que se encuentra en la mitad).
  - Todos los elementos menores al pivote se colocan en un array y los mayores en otro.
  - Se aplica el mismo procedimiento de forma ***recursiva***, sobre los subarrays hasta que solo exista un elemento.
- La complejidad del algoritmo es O(n·logn).

Método de Ordenación: Quicksort

- División del array

44	55	12	42	94	18	06	67
44	55	12	42	94	18	06	67
06	55	12	42	94	18	44	67
06	18	12	42	94	55	44	67
06	18	12	42	94	55	44	67

Método de Ordenación: Quicksort

06	18	12	42	94	55	44	67
06	12	18	42	44	55	94	67
06	12	18	42	44	55	94	67
06	12	18	42	44	55	67	94

Método de Ordenación: Quicksort

```
public class OrdQuicksort {
  public static void main(String args[]) {
     double data[]={321,123,213,234,1,4,5,6}; //Array a ordenar

    //Invocacion metodo ordenacion
     quicksort(data, 0, data.length-1);

     //Imprime el contenido del array ordenado
     for (int i = 0; i < data.length; i++) {
       System.out.println(data[i]);
     }
  }
```

Método de Ordenación: Quicksort

```
  static void quicksort(double data[], int izq, int der) {
     int i = izq, j = der;
     double pivote = data[izq + (der-izq)/2]; //elem. pivote (mitad)
     // Division en dos subarrays
     while (i <= j) {
       while (data[i] < pivote)  i++; //valores menores al pivote
       while (data[j] > pivote)  j--; //valores mayores al pivote
       if (i <= j) {  // intercambiar y seguir
         double tmp = data[i]; data[i] = data[j]; data[j] = tmp;
         i++; j--; }
     }
     if (izq < j)  quicksort(data, izq, j); // Recursion subarray <
     if (i < der)  quicksort(data, i, der); // Recursion subarray >
  }
}
```

Algoritmos comunes - Búsqueda

- Búsqueda lineal o secuencial
  - Se aplica a arrays desordenados.
  - La complejidad del algoritmo es O(n).
- Búsqueda binaria
  - Se aplica a arrays ordenados.
  - Compara el elemento en la mitad del array con el buscado, si es menor excluye la mitad menor, si es mayor excluye la mitad mayor.
  - Repetir hasta encontrar el valor buscado o no se puede dividir.

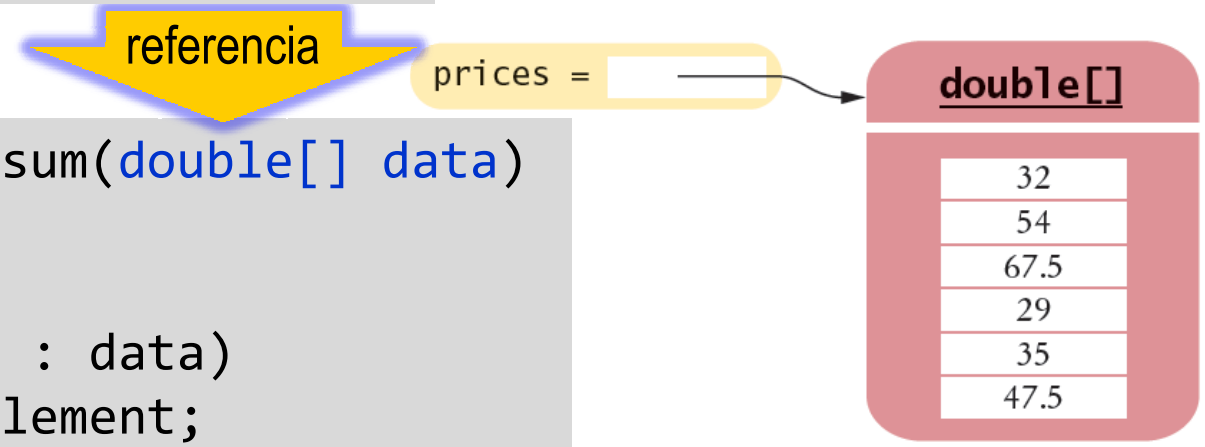
Búsqueda binaria

```
double searchedValue = XXX;  // Valor a buscar
boolean found = false; int low = 0, pos = 0;
int high = data.length - 1;
while (low <= high && !found) {
  pos = (low + high) / 2;  // Mitad del array
  if (data[pos] == searchedValue)
  { found = true; }      // Encontrado
  else if (data[pos] < searchedValue)
  { low = pos + 1; }     // Busca en la primera mitad
  else { high = pos - 1; } // Busca en la segunda mitad
}
if (found)
{ System.out.println("Encontrado en la posicion " + pos+1); }
else
{ System.out.println("No encontrado"); }
```

Paso de arrays a métodos

- Es común usar arrays como parámetros de métodos y como valor de retorno de métodos.
  - Los arrays se pasan como *referencia* en los métodos.
- Ej.: método para sumar los elementos de un array.

```
precioTotal = sum(prices);
```




```
public static double sum(double[] data)
{
  double total = 0;
  for (double element : data)
    total = total + element;
  return total;
}
```

Paso de referencias

- El paso de una referencia da al método invocado acceso a todos los elementos.
  - Puede modificar los datos.
- Ej.: método que multiplica los elementos de un array por un valor.

```
multiply(  values,          10);
```

reference

value

```
public static void multiply(double[] data, double factor)
{
  for (int i = 0; i < data.length; i++)
    data[i] = data[i] * factor;
}
```

Listas

- Cuando se escribe un programa que colecciona datos, no siempre se sabe cuántos valores se tendrá.
- En tal caso una lista ofrece dos ventajas significativas:
  - La lista puede crecer o disminuir como sea necesario.
  - La clase ArrayList ofrece métodos para las operaciones comunes, tal como insertar o eliminar elementos.
- Las listas con una *clase genérica* (puede contener muchos tipos de objetos) que se encuentra en el paquete `java.util.ArrayList`

Uso de Listas

- Durante la declaración se indica el tipo de los elementos.
  - Dentro de `< >` como el tipo de “parámetro”
  - El tipo debe ser una clase
  - No se puede usar tipos de datos primitivos (int, double…)

```
ArrayList<String> names = new ArrayList<String>();
```

- Métodos útiles de `ArrayList`
  - add: añade un elemento
  - get: retorna un elemento
  - remove: elimina un elemento
  - set: cambia un elemento
  - size: longitud del array

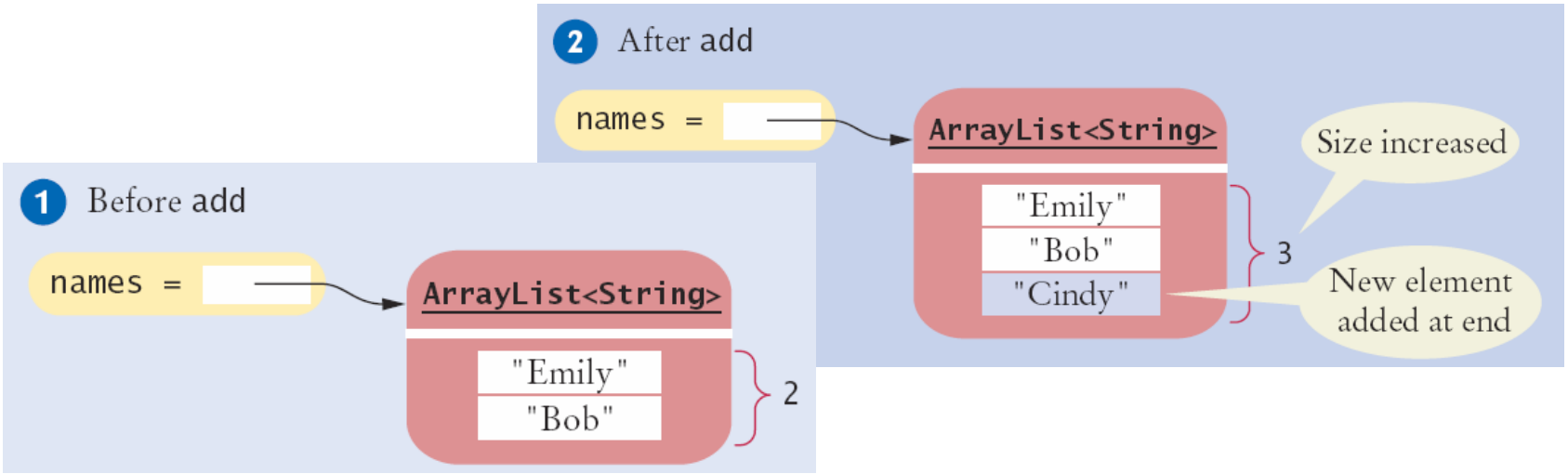
Inserción de un elemento con add()

- El método `add` tiene dos versiones:
  - Pasar un elemento para añadirse al final

```
names.add("Cindy");
```

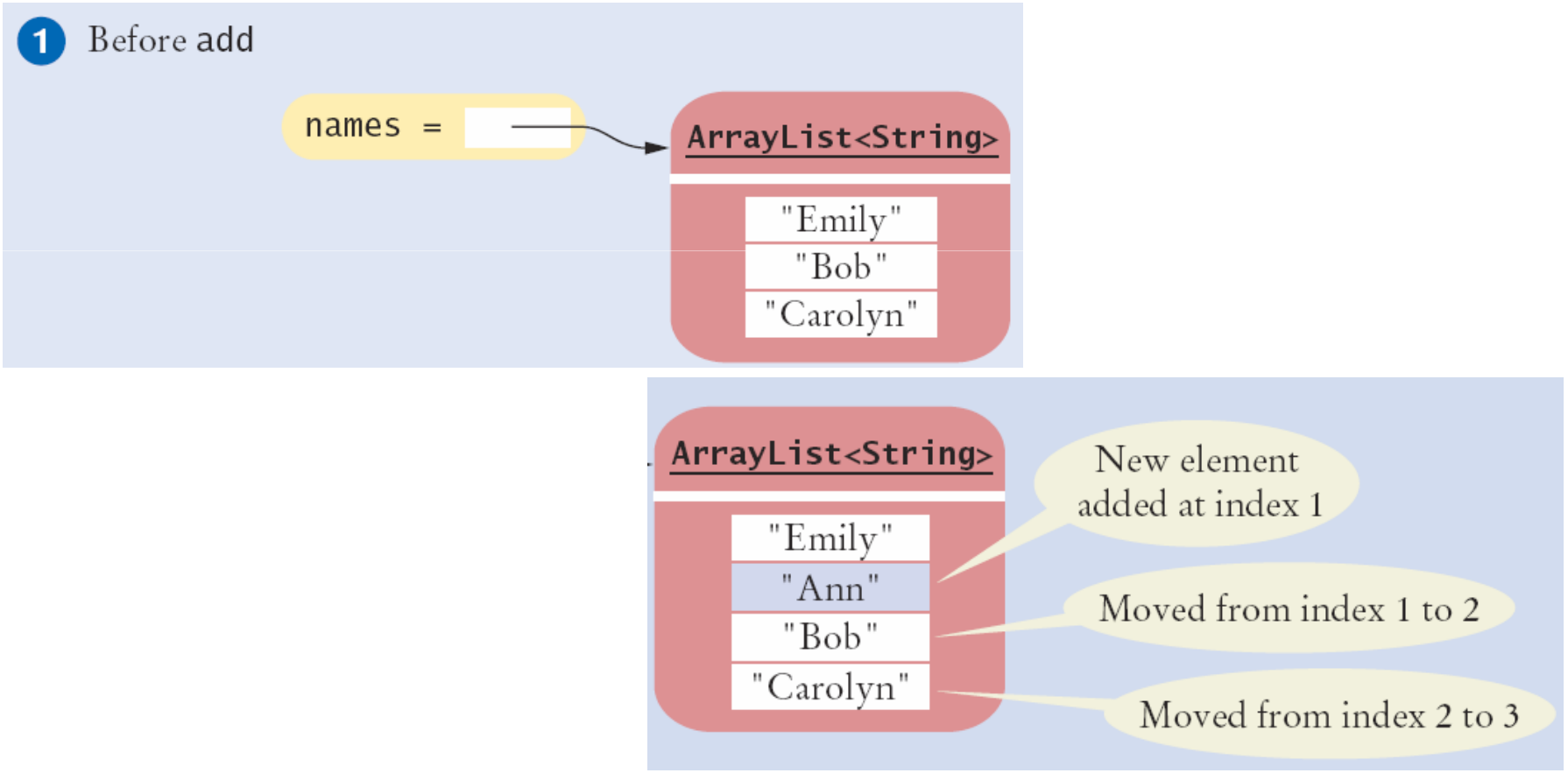
  - Pasar un índice y el nuevo elemento a añadir en esa posición. Los otros elementos se mueven.

```
names.add(1, "Cindy");
```



Inserción de un elemento con add()

```
names.add(1, "Ann");
```

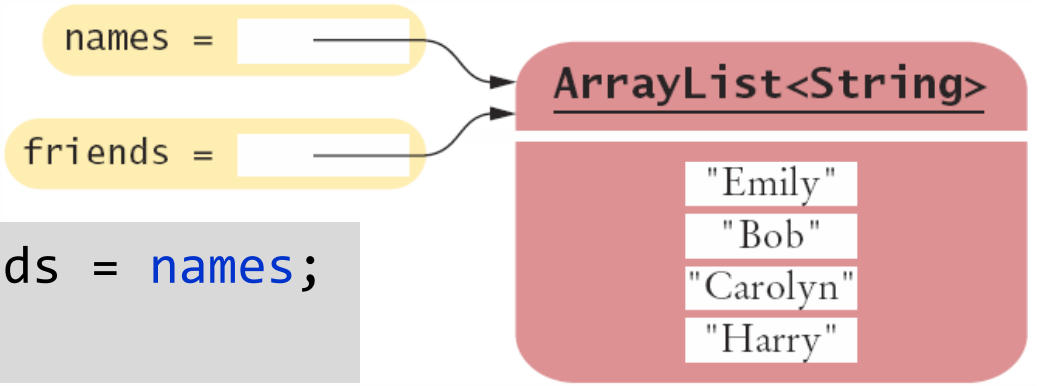


Uso de ArrayList

```
ArrayList<String> names = new ArrayList<String>();

names.add("Ann");
names.add("Cindy");

System.out.println(names);

names.add(1,"Bob");

names.remove(0);

names.set(0, "Bill");

String name = names.get(i);
String last = names.get(names.size() - 1);
```

Copia de ArrayList

- `ArrayList` mantiene una referencia como los arrays.
- Copiando una referencia:




```
ArrayList<String> friends = names;
friends.add("Harry");
```

- Para hacer una copia, pasar la referencia del ArrayList original al constructor del nuevo:

referencia

```
ArrayList<String> newNames = new ArrayList<String>(names);
```

ArrayList y métodos

- De igual manera que los arrays, un `ArrayList` puede ser usado como parámetro o valor retornado.
- Ejemplo: método que recibe un ArrayList y devuelve la lista invertida

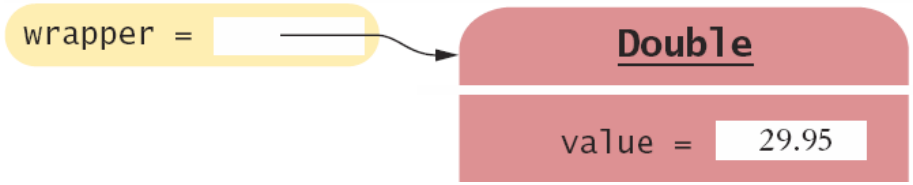
referencia

```
public static ArrayList<String> reverse(ArrayList<String> names)
{
 // Crea una lista para el resultado del metodo
  ArrayList<String> result = new ArrayList<String>();
 // Recorre la lista de nombres en orden inverso (último a primero)
  for (int i = names.size() - 1; i >= 0; i--)
  {
    // Añade cada nombre al resultado
    result.add(names.get(i));
  }
  return result;
}
```

Wrappers y auto-boxing

- Java ofrece las clases wrapper para tipos primitivos.
  - Las conversiones son automáticas usando ***auto-boxing***
  - Tipo primitivo a clase Wrapper

```
double x = 29.95;
Double wrapper;
wrapper = x;  // boxing
```



  - Clase Wrapper a tipo primitivo

```
double x;
Double wrapper = 29.95;
x = wrapper;  // unboxing
```

Primitive Type	Wrapper Class
byte	Byte
boolean	Boolean
char	Character
double	Double
float	Float
int	Integer
long	Long
short	Short

Wrappers y auto-boxing

- No se puede usar tipos primitivos en un ArrayList, pero se puede usar sus clases wrapper.
  - Depende del auto-boxing para la conversión
- Declarar el ArrayList con clases wrapper para los tipos primitivos

```
double x = 19.95;
ArrayList<Double> data = new ArrayList<Double>();
data.add(29.95);              // boxing
data.add(x);                  // boxing
double x = data.get(0);       // unboxing
```

Algoritmos con ArrayList

- La conversión de arrays a ArrayList requiere el cambio de:
  - uso de índices `[i]`
  - `data.length`
- a
  - métodos `get()`
  - `data.size()`

```
double largest = data[0];
for (int i = 1; i < data.length; i++)
{
   if (data[i] > largest) {
      largest = data[i];
   }
}
```

```
double largest = data.get(0);
for (int i = 1; i < data.size(); i++)
{
   if (data.get(i) > largest) {
      largest = data.get(i);
   }
}
```

Cuándo usar Arrays o ArrayList

- Usar arrays si:
  - el tamaño del array nunca cambia
  - se tiene una lista grande de tipos primitivos
  - lo pide el “jefe”
- Usar un ArrayList
  - en cualquiera de los otros casos
  - especialmente si se tiene un número desconocido de valores de entrada

Cuidado con length o size

- No hay consistencia para determinar el número de elementos en un Array, ArrayList o String

Data Type	Number of Elements
Array	`a.length`
Array list	`a.size()`
`String`	`a.length()`