Capítulo 7

Matrices unidimensionales

Problema de apertura

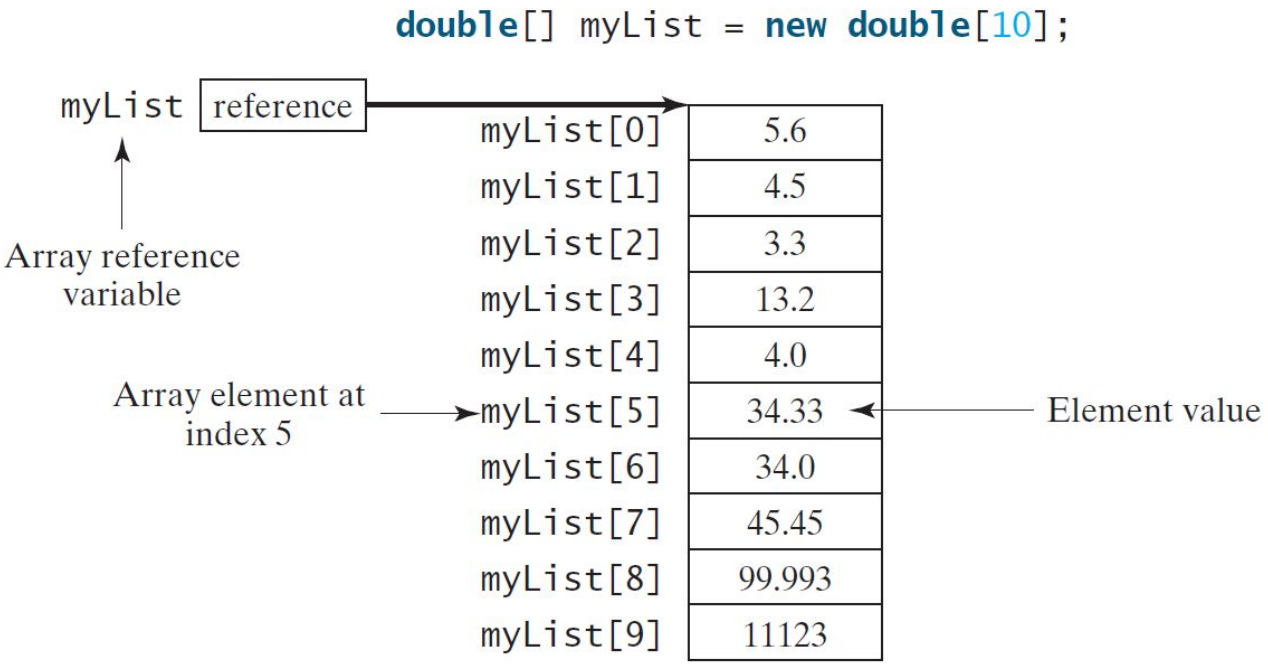
Lea cien números, calcule su promedio o media y descubra cuántos números están por encima de la media.

Objetivos

- Describir por qué las matrices son necesarias en la programación .
- Declarar variables de referencia de matriz y crear matrices .
- Obtener el tamaño de la matriz usando arrayRefVar.length y conocer los valores por defecto en una matriz.
- Acceder a los elementos de la matriz mediante índices.
- Declarar, crear e inicializar una matriz usando un inicializador de matriz.
- Programar operaciones de matriz comunes (mostrar matrices, sumar todos los elementos, encontrar los elementos mínimo y máximo, mezcla aleatoria y elementos de desplazamiento).
- Simplificar la programación utilizando los bucles for-each.
- Aplicar matrices en el desarrollo de aplicaciones.
- Copiar contenidos de una matriz a otra.
- Desarrollar e invocar métodos con argumentos de matriz y valores de retorno.
- Definir un método con una lista de argumentos de longitud variable.
- Buscar elementos usando el algoritmo de búsqueda lineal o binario.
- Ordenar una matriz usando el enfoque de selección de clasificación.
- Usar los métodos en la clase java.util.Arrays.
- Argumentos al método principal desde la línea de comando

Definición de matrices

Matriz, array en inglés, es una estructura de datos que representa una colección de los mismos tipos de datos.



Declarando variables de matriz

El tipo de la tabla puede ser cualquier tipo válido en Java. Se admiten dos sintaxis para la declaración de una tabla en Java:

- `tipodato[] arrayRefVar;`

  Ejemplo:

  ```
  double[] myList;
  ```

- `tipodato arrayRefVar[];` // Este estilo está permitido, pero es más habitual el anterior

  Ejemplo:

  ```
  double myList[];
  ```

Creando Matrices

Hemos declarado la tabla pero habrá que instanciarla dándole el número de elementos para que se reserve el espacio de memoria necesario.

Para crear la tabla se usa la sentencia new de la siguiente forma:

```
arrayRefVar = new tipodato[tamañoArray];
```

Ejemplo:

```
myList = new double[10];
myList[0] referencia al primer elemento del array.
myList[9] referencia al último elemento del array.
```

Declarando y Creando a la vez

La inicialización o instanciación de la tabla se puede hacer también en la misma sentencia donde se declara:

- ```
  tipodato[] arrayRefVar = new tipodato[tamañoArray];
  ```

  ```
  double[] myList = new double[10];
  ```

- ```
  tipodato arrayRefVar[] = new tipodato[tamañoArray];
  ```

  ```
  double myList[] = new double[10];
  ```

En Java, una tabla no puede cambiar de tamaño, es decir, una vez instanciada, va a tener el mismo número de elementos hasta que sea destruida.

Valores predeterminados

Cuando se crea un array, a sus elementos se les asigna el valor predeterminado de

- 0 para los tipos de datos primitivos numéricos,
- '\u0000' para tipos char, y
- false para tipos boolean.

Declarar, crear, inicializar

Declarando, creando, inicializando en un solo paso:

tipo[] nombre_tabla = {valor0, valor1, valor2, ...};

```
double[] myList = {1.9, 2.9, 3.4, 3.5};
```

Esta notación abreviada es equivalente a las siguientes afirmaciones:

```
double[] myList = new double[4];
myList[0] = 1.9;
myList[1] = 2.9;
myList[2] = 3.4;
myList[3] = 3.5;
```

ATENCIÓN

Usando la notación abreviada, debe declarar, crear e inicializar el array, todo en una declaración. Dividirlo causaría un error de sintaxis.

Por ejemplo, lo siguiente está mal :

```
double[] myList;

myList = {1.9, 2.9, 3.4, 3.5};
```

La longitud de un array

Una vez que se crea un array, su tamaño es fijo. No puede ser cambiado. Puedes encontrar su tamaño usando

arrayRefVar.length

Por ejemplo,

myList.length devuelve 10

Recorrido de tablas

Para realizar un recorrido por todos los elementos de una tabla mediante un índice, se suele usar un bucle for de la siguiente forma:

```
for (int i = 0; i < tabla.length; i++) {
procesar(tabla[i]);
}
```

Variables indexadas

A los elementos del array se accede a través del índice. Los índices del array comienza desde 0 hasta arrayRefVar.length-1.

Cada elemento de la matriz se representa con la siguiente sintaxis, conocida como *variable indexada*:

```
arrayRefVar[indice];
```

Después de crear un array, se puede usar una variable indexada de la misma manera que una variable regular. Por ejemplo, el siguiente código suma el valor de myList[0] y myList[1] en myList[2]

```
myList[2] = myList[0] + myList[1];
```

Seguir un programa con matrices

Declarar valores de las variables de array, crear un array, y asignar su referencia a los valores

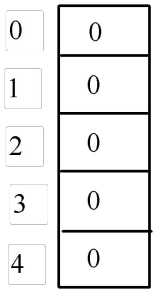
```
public class Test {
  public static void main(String[] args) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

Seguir un programa con matrices

i se convierte en1

```
public class Test {
  public static void main(String[] args) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

After the array is created

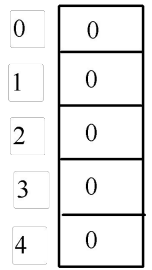


Seguir un programa con matrices

i (=1) es menor que 5

```
public class Test {
  public static void main(String[] args) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

After the array is created

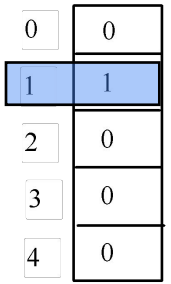


Seguir un programa con matrices

Después de ejecutar esta línea, value[1] es 1

```
public class Test {
  public static void main(Stri      gs) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

After the first iteration



Seguir un programa con matrices

```
public class Test {
  public static void main(String[] a
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

Después de i++, i es 2

After the first iteration

0	0
1	1
2	0
3	0
4	0

Seguir un programa con matrices

```
public class Test {
  public static void main(String[] args) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

i (= 2) es menor que 5

After the first iteration

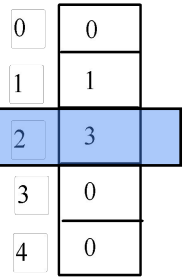
0	0
1	1
2	0
3	0
4	0

Seguir un programa con matrices

Después de ejecutar esta línea, values[2] es 3 (2 + 1)

```
public class Test {
  public static void main(String[] args) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

After the second iteration



Seguir un programa con matrices

Después de esto, i es 3.

```
public class Test {
  public static void main(S      [] args) {
    int[] values = new int[5
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

After the second iteration

0	0
1	1
2	3
3	0
4	0

Seguir un programa con matrices

```
public class Test {
  public static void main(String[] args) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

i (=3) sigue siendo menos de 5.

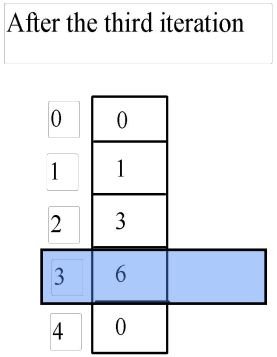
After the second iteration

0	0
1	1
2	3
3	0
4	0

Seguir un programa con matrices

Después de esta línea, values[3] se convierten en 6 (3 + 3)

```
public class Test {
  public static void main(String[]
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```



Seguir un programa con matrices

```
public class Test {
  public static void main(String[] args) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

Después de esto, i es 4

After the third iteration

0	0
1	1
2	3
3	6
4	0

Seguir un programa con matrices

i (=4) sigue siendo menos de 5.

```
public class Test {
  public static void main(String[] args) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

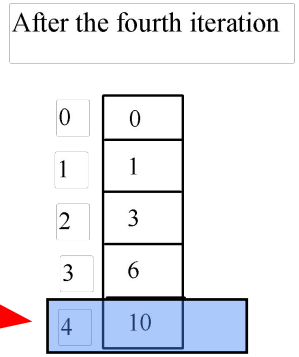
After the third iteration

0	0
1	1
2	3
3	6
4	0

Seguir un programa con matrices

Después de esto, values[4] es 10 (4 + 6)

```
public class Test {
  public static void main(Stri    rgs) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```



Seguir un programa con matrices

```
public class Test {
  public static void main(String[] ar
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

Después de i++, i es 5

After the fourth iteration

0	0
1	1
2	3
3	6
4	10

Seguir un programa con matrices

i (=5) < 5 es falso. Sale del bucle

```
public class Test {
  public static void main(String[] args) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

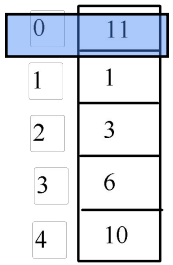
After the fourth iteration

0	0
1	1
2	3
3	6
4	10

Seguir un programa con matrices

Después de esta línea, values[0] es 11 (1 + 10)

```
public class Test {
  public static void main(Strin      gs) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```



Procesamiento habitual en Arrays

1. (Inicializando arrays desde teclado)
2. (Inicializando arrays con valores aleatorios)
3. (Impresión de arrays)
4. (Sumando todos los elementos)
5. (Encontrar el elemento más grande)
6. (Encontrar el índice más pequeño del elemento más grande)
7. (Mezclar aleatoriamente)
8. (Desplazando Elementos)

Veamos ejemplos en las siguientes diapositivas.

Inicializando arrays desde teclado

```
java.util.Scanner sc = new java.util.Scanner(System.in);
System.out.print("Enter " + myList.length + " values: ");
for (int i = 0; i < myList.length; i++)
  myList[i] = sc.nextDouble();
```

Inicializando arrays con valores aleatorios

```
for (int i = 0; i < myList.length; i++) {
  myList[i] = Math.random() * 100;
}
```

Impresión de arrays

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

Sumando todos los elementos del array

```
double total = 0;
for (int i = 0; i < myList.length; i++) {
  total += myList[i];
}
```

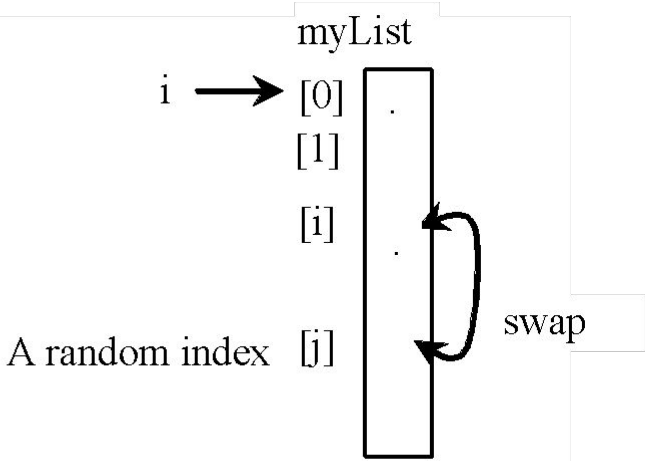
Encontrar el elemento más grande

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

Mezclar aleatoriamente

```
for (int i = 0; i < myList.length - 1; i++) {
  // Genera un índice j al azar
  int j = (int)(Math.random()
    * myList.length);

  // Cambia myList [i] con myList [j]
  double temp = myList[i];
  myList[i] = myList[j];
  myList[j] = temp;
}
```



Desplazando Elementos

```
double temp = myList[0]; // Retener el primer elemento

// Desplazar elementos restantes
for (int i = 1; i < myList.length; i++) {
        myList[i - 1] = myList[i];
}

// Mover el primer elemento para rellenar la última posición
myList[myList.length - 1] = temp;
```

Bucle for mejorado (bucle for-each)

JDK 1.5 introdujo un nuevo bucle for que le permite recorrer la matriz completa secuencialmente sin usar una variable de índice. Por ejemplo, el siguiente código muestra todos los elementos en la matriz myList:

```
for (double value: myList)
  System.out.println(value);
```

Es como decir: "ve sacando uno a uno los elementos del array myList y deposita cada uno de esos elementos en la variable value que es de tipo double".

En general, la sintaxis es

```
for (elementType value: arrayRefVar) {
  // Process the value
}
```

Con este bucle se recorren los elementos de una tabla mediante una variable (elemento) que va tomando los valores de todos los elementos de la tabla desde el

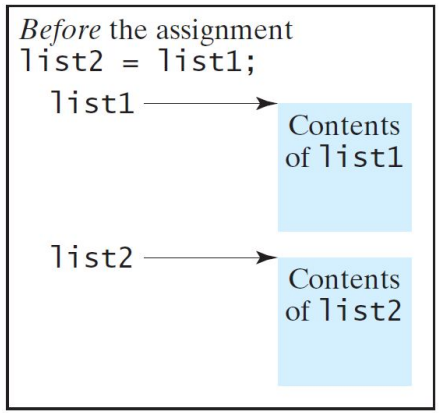
primero hasta el último.

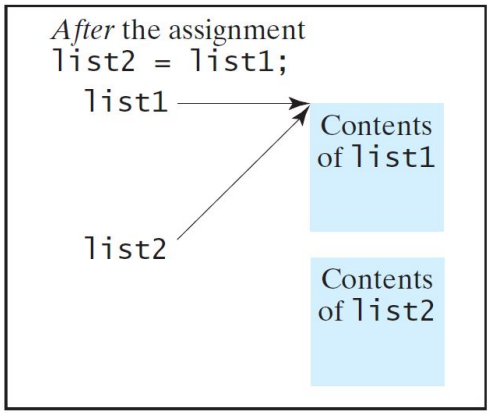
Aun así, tiene que usar una variable índice si desea recorrer la matriz en un orden diferente o cambiar los elementos en la matriz.

Copiar Arrays

A menudo, en un programa, necesita duplicar una matriz o una parte de una matriz. En tales casos, podría intentar usar la declaración de asignación (=), de la siguiente manera :

list2 = list1;





En Java, puede usar instrucciones de asignación para copiar variables de tipo de datos primitivos, pero no matrices. Asignar una variable de matriz a otra variable de matriz en realidad copia una referencia a otra y hace que ambas variables apunten a la misma ubicación de memoria.

Copiar Arrays

Podemos asignar el valor de una variable tipo tabla a otra del mismo tipo. OJO! en ese caso, lo que realmente se asigna es la referencia a la tabla.

Para asignar el contenido de una tabla a otra tienes que escribir un bucle que copie cada elemento del array origen al elemento correspondiente en el array destino

```
int[] sourceArray = {2, 3, 1, 5, 10};
int[] targetArray = new int[sourceArray.length];

for (int i = 0; i < sourceArrays.length; i++)
   targetArray[i] = sourceArray[i];
```

La utilidad arraycopy

Otra opción es utilizar el método *arraycopy* en la clase java.lang.System para copiar arrays en lugar de utilizar un bucle. La sintaxis para *arraycopy* es:

```
arraycopy(sourceArray, src_pos, targetArray, tar_pos, length);
```

Ejemplo:

```
System.arraycopy(sourceArray, 0, targetArray, 0,
   sourceArray.length);
```

Los parámetros srcPos y tarPos indican las posiciones de inicio en sourceArray y TargetArray, respectivamente. Length indica el nº de elementos

El método arraycopy no asigna espacio de memoria para la matriz objetivo. La matriz de destino ya debe haber sido creada con su espacio de memoria asignado. Después de que se realiza la copia, targetArray y sourceArray tienen el mismo contenido pero ubicaciones de memoria independientes.

Problemas

1.- ANALIZAR NÚMEROS: Lea cien números, calcule su media y descubra cuántos números están por encima de la media.

2.- Dado un array, imprime todos sus elementos en una sola línea, separados por espacios.

3.- Escribe un programa que encuentre el valor más grande y el valor más pequeño de un array de enteros.

4.- Escribe un programa que encuentre el índice del primer elemento más grande en un array.

5.- Escribe un programa que desplace todos los elementos de un array una posición a la derecha. El último elemento debe moverse a la primera posición.

6.- Escribe un programa que copie manualmente todos los elementos de un array en otro array. Haz una nueva versión utilizando el método correspondiente.

Pasar matrices a métodos

Del mismo modo que puede pasar valores de tipo primitivo a métodos, también puede pasar arrays a métodos.

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

Llama al metodo

```
int[] list = {3, 1, 2, 6, 4, 2};
printArray(list);
```

Llamada al metodo

```
printArray(new int[]{3, 1, 2, 6, 4, 2});
```

Array anónimo

Matriz anónima

La sentencia

```
printArray(new int[]{3, 1, 2, 6, 4, 2});
```

crea un array usando la siguiente sintaxis:

```
new dataType[]{literal0, literal1, ..., literalk};
```

No hay una variable de referencia explícita para la matriz. Dicha matriz se llama una matriz anónima.

Paso por valor

Java utiliza paso por valor para pasar argumentos a un método. Existen importantes diferencias entre pasar un valor de variables de tipos de datos primitivos y pasar arrays.

- Para un parámetro de un valor de tipo primitivo, se pasa el valor real. Cambiar el valor del parámetro local dentro del método no afecta el valor de la variable fuera del método.

- Para un parámetro de un tipo array, el valor del parámetro contiene una referencia a un array; esta referencia se pasa al método. Cualquier cambio en el array que ocurra dentro del cuerpo del método afectará al array original que se pasó como argumento.

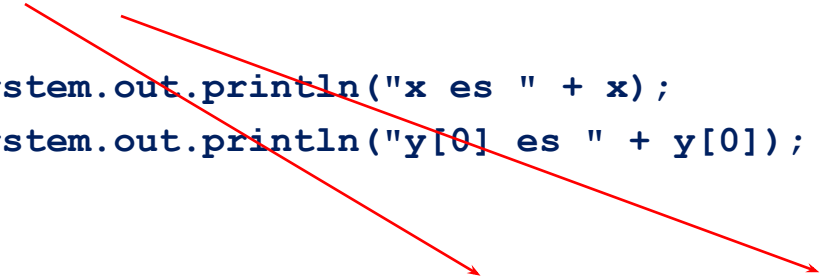
Ejemplo Simple

```
public class Test {
  public static void main(String[] args) {
    int x = 1; // x representa un valor entero
    int[] y = new int[10]; // y representa un array de enteros

    m(x, y); // llama a m con los argumentos x e y

    System.out.println("x es " + x);
    System.out.println("y[0] es " + y[0]);
  }

  public static void m(int number, int[] numbers) {
    number = 1001; // Asigna un nuevo valor a number
    numbers[0] = 5555; // Asigna un nuevo valor a numbers[0]
  }
}
```

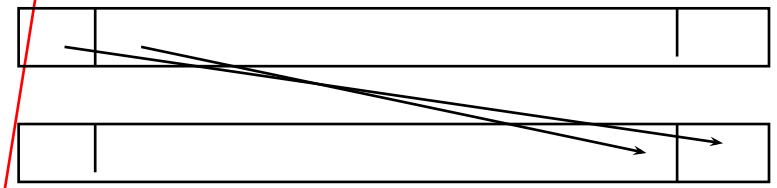


Devolviendo un array desde un método

```
public static int[] reverse(int[] list) {
  int[] result = new int[list.length];

  for (int i = 0, j = result.length - 1;
       i < list.length; i++, j--) {
    result[j] = list[i];
  }

  return result;
}
```




```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

Seguimiento del método reverse

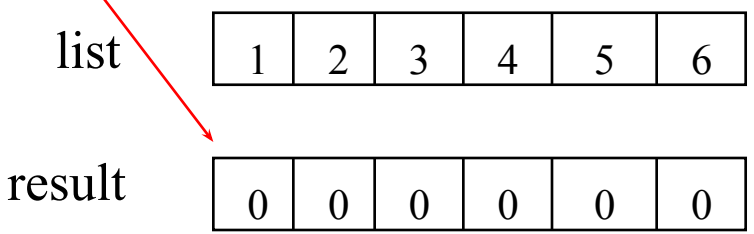
```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);

  public static int[] reverse(int[] list) {
    int[] result = new int[list.length];

    for (int i = 0, j = result.length - 1;
         i < list.length; i++, j--) {
      result[j] = list[i];
    }

    return result;
  }
```

Declara y crea el array result



Seguimiento del método reverse

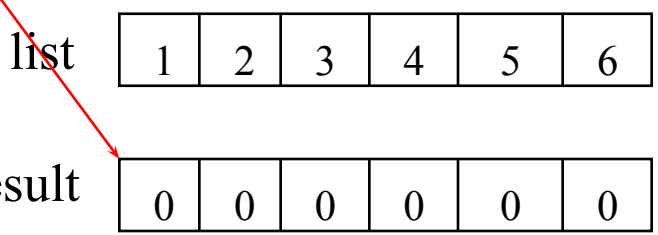
```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);

  public static int[] reverse(int[] list) {
    int[] result = new int[list.length];

    for (int i = 0, j = result.length - 1;
         i < list.length; i++, j--) {
      result[j] = list[i];
    }

    return result;
  }
```

Inicializa i = 0 y j = 5



Seguimiento del método reverse

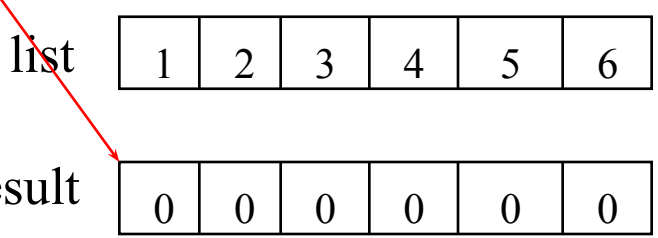
```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);

  public static int[] reverse(int[] list) {
    int[] result = new int[list.length];

    for (int i = 0, j = result.length - 1;
         i < list.length; i++, j--) {
      result[j] = list[i];
    }

    return result;
  }
```

i (= 0) es menor que 6



Seguimiento del método reverse

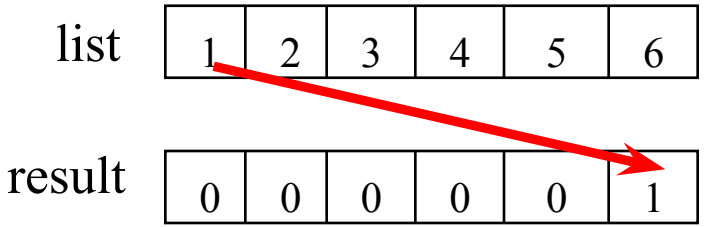
```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);

  public static int[] reverse(int[] list) {
    int[] result = new int[list.length];

    for (int i = 0, j = result.length - 1;
         i < list.length; i++, j--) {
      result[j] = list[i];
    }

    return result;
  }
```

i = 0 y j = 5
Asigna list[0] a result[5]



Seguimiento del método reverse

```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);

  public static int[] reverse(int[] list) {
    int[] result = new int[list.length];

    for (int i = 0, j = result.length - 1;
         i < list.length; i++, j--) {
      result[j] = list[i];
    }

    return result;
  }
```

Después de esto, i es 1 y j es 4

list	1	2	3	4	5	6

result	0	0	0	0	0	1

Seguimiento del método reverse

```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);

  public static int[] reverse(int[] list) {
    int[] result = new int[list.length];

    for (int i = 0, j = result.length - 1;
         i < list.length; i++, j--) {
      result[j] = list[i];
    }

    return result;
  }
```

i (=1) es menor que 6

list	1	2	3	4	5	6

result	0	0	0	0	0	1

Seguimiento del método reverse

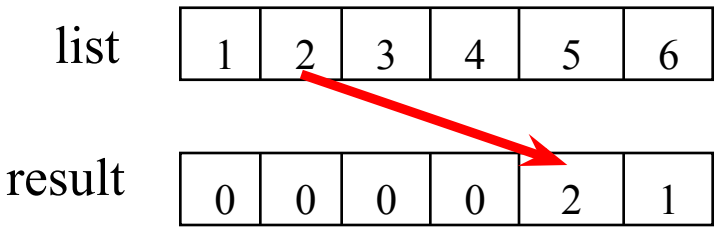
```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);

  public static int[] reverse(int[] list) {
    int[] result = new int[list.length];

    for (int i = 0, j = result.length - 1;
         i < list.length; i++, j--) {
      result[j] = list[i];
    }

    return result;
  }
```

i = 1 y j = 4
Asigna list[1] a result[4]



Seguimiento del método reverse

```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);

  public static int[] reverse(int[] list) {
    int[] result = new int[list.length];

    for (int i = 0, j = result.length - 1;
         i < list.length; i++, j--) {
      result[j] = list[i];
    }

    return result;
  }
```

Después de esto, i es 2 y j es 3

list	1	2	3	4	5	6

result	0	0	0	0	2	1

Seguimiento del método reverse

```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);

  public static int[] reverse(int[] list) {
    int[] result = new int[list.length];

    for (int i = 0, j = result.length - 1;
         i < list.length; i++, j--) {
      result[j] = list[i];
    }

    return result;
  }
```

i (=2) aun es menor que 6

list	1	2	3	4	5	6

result	0	0	0	0	2	1

Seguimiento del método reverse

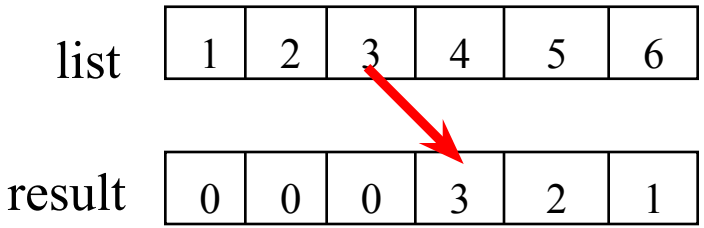
```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);

  public static int[] reverse(int[] list) {
    int[] result = new int[list.length];

    for (int i = 0, j = result.length - 1;
         i < list.length; i++, j--) {
      result[j] = list[i];
    }

    return result;
  }
```

i = 2 y j = 3
Asigna list[i] a result[j]



Seguimiento del método reverse

```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);

  public static int[] reverse(int[] list) {
    int[] result = new int[list.length];

    for (int i = 0, j = result.length - 1;
         i < list.length; i++, j--) {
      result[j] = list[i];
    }

    return result;
  }
```

Después de esto, i es 3 y j es 2

list	1	2	3	4	5	6

result	0	0	0	3	2	1

Seguimiento del método reverse

```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);

  public static int[] reverse(int[] list) {
    int[] result = new int[list.length];

    for (int i = 0, j = result.length - 1;
         i < list.length; i++, j--) {
      result[j] = list[i];
    }

    return result;
  }
```

i (=3) aun es menor que 6

list	1	2	3	4	5	6

result	0	0	0	3	2	1

Seguimiento del método reverse

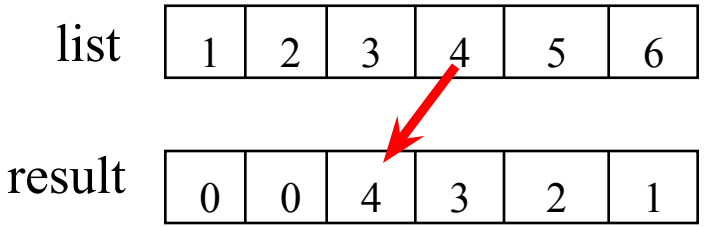
```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);

  public static int[] reverse(int[] list) {
    int[] result = new int[list.length];

    for (int i = 0, j = result.length - 1;
         i < list.length; i++, j--) {
      result[j] = list[i];
    }

    return result;
  }
```

i = 3 y j = 2
Asigna list[i] a result[j]



Seguimiento del método reverse

```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);

  public static int[] reverse(int[] list) {
    int[] result = new int[list.length];

    for (int i = 0, j = result.length - 1;
         i < list.length; i++, j--) {
      result[j] = list[i];
    }

    return result;
  }
```

Después de esto, i es 4 y j es 1

list	1	2	3	4	5	6

result	0	0	4	3	2	1

Seguimiento del método reverse

```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);

  public static int[] reverse(int[] list) {
    int[] result = new int[list.length];

    for (int i = 0, j = result.length - 1;
         i < list.length; i++, j--) {
      result[j] = list[i];
    }

    return result;
  }
```

i (=4) aun es menor que 6

list

1	2	3	4	5	6

result

0	0	4	3	2	1

Seguimiento del método reverse

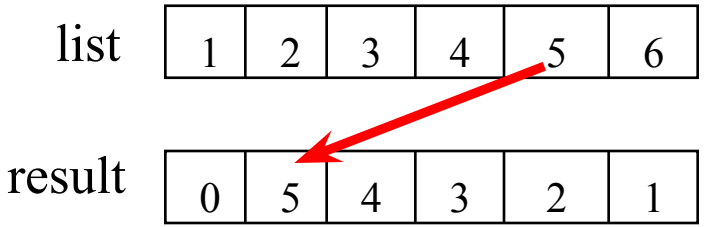
```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);

  public static int[] reverse(int[] list) {
    int[] result = new int[list.length];

    for (int i = 0, j = result.length - 1;
         i < list.length; i++, j--) {
      result[j] = list[i];
    }

    return result;
  }
```

i = 4 y j = 1
Asigna list[i] a result[j]



Seguimiento del método reverse

```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);

  public static int[] reverse(int[] list) {
    int[] result = new int[list.length];

    for (int i = 0, j = result.length - 1;
         i < list.length; i++, j--) {
      result[j] = list[i];
    }

    return result;
  }
```

Después de esto, i es 5 y j es 0

list	1	2	3	4	5	6

result	0	5	4	3	2	1

Seguimiento del método reverse

```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);

  public static int[] reverse(int[] list) {
    int[] result = new int[list.length];

    for (int i = 0, j = result.length - 1;
         i < list.length; i++, j--) {
      result[j] = list[i];
    }

    return result;
  }
```

i (=5) aun es menor 6

list	1	2	3	4	5	6

result	0	5	4	3	2	1

Seguimiento del método reverse

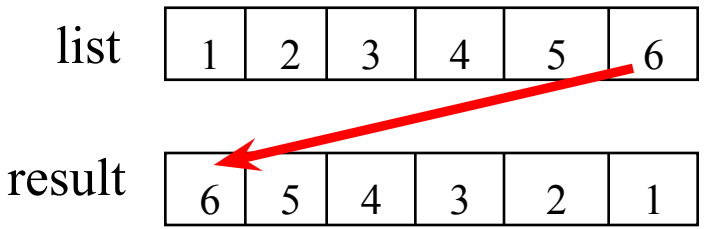
```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);

  public static int[] reverse(int[] list) {
    int[] result = new int[list.length];

    for (int i = 0, j = result.length - 1;
         i < list.length; i++, j--) {
      result[j] = list[i];
    }

    return result;
  }
```

i = 5 y j = 0
Asigna list[i] a result[j]



Seguimiento del método reverse

```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);

  public static int[] reverse(int[] list) {
    int[] result = new int[list.length];

    for (int i = 0, j = result.length - 1;
         i < list.length; i++, j--) {
      result[j] = list[i];
    }

    return result;
  }
```

Después de esto, i es 6 y j es -1

list	1	2	3	4	5	6

result	6	5	4	3	2	1

Seguimiento del método reverse

```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);

  public static int[] reverse(int[] list) {
    int[] result = new int[list.length];

    for (int i = 0, j = result.length - 1;
         i < list.length; i++, j--) {
      result[j] = list[i];
    }

    return result;
  }
```

i (=6) < 6 es falso. Así que sale del bucle.

list	1	2	3	4	5	6

result	6	5	4	3	2	1

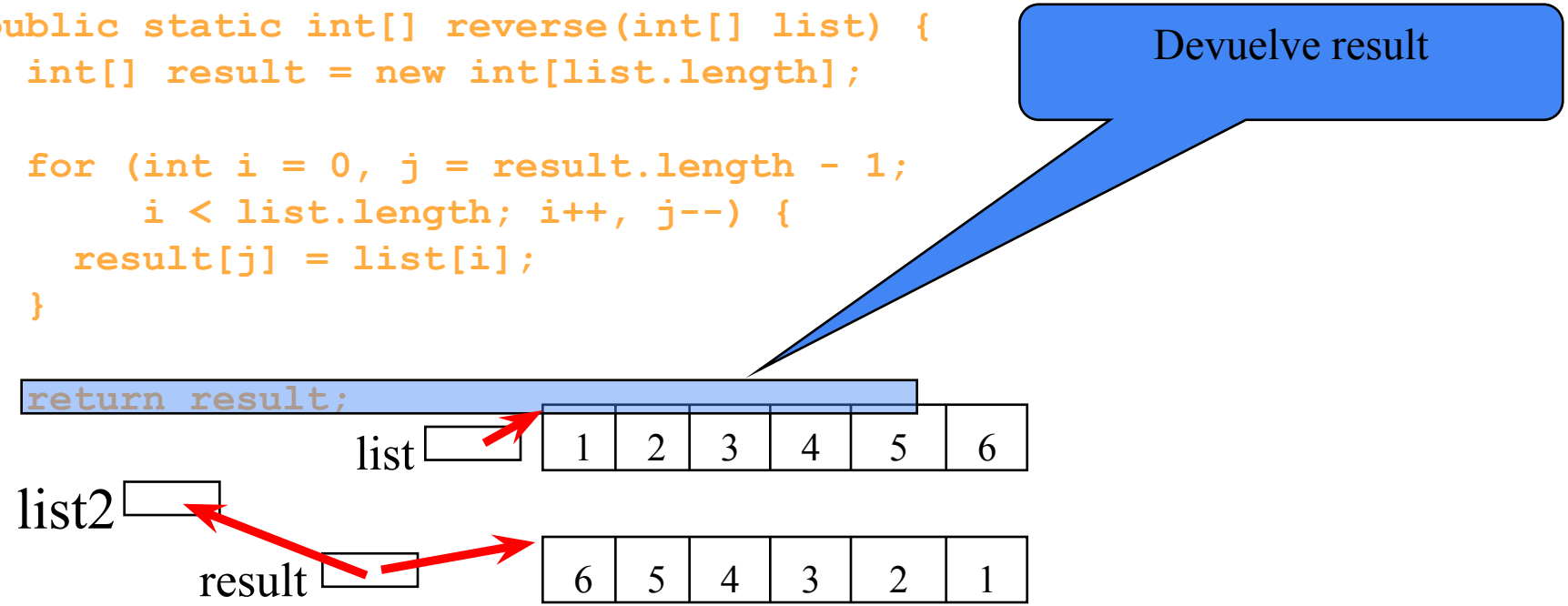
Seguimiento del método reverse

```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);

  public static int[] reverse(int[] list) {
    int[] result = new int[list.length];

    for (int i = 0, j = result.length - 1;
         i < list.length; i++, j--) {
      result[j] = list[i];
    }

    return result;
  }
```



Cantidad variable de Arguments

Podemos pasar una cantidad variable de argumentos del mismo tipo a un método. El parámetro en el método se declara de la siguiente manera:

nombreTipo... nombreParametros

En la declaración de método, especifica el tipo seguido de puntos suspensivos (...). Solo se puede especificar un parámetro de longitud variable en un método, y este parámetro debe ser el último parámetro. Cualquier parámetro regular debe precederlo.

Java trata un parámetro de longitud variable como una matriz. Puede pasar una matriz o una cantidad variable de argumentos a un parámetro de longitud variable. Al invocar un método con un número variable de argumentos, Java crea una matriz y le pasa los argumentos.

Clase Arrays. Funciones predefinidas

En la API estándar Java podemos encontrar la clase Arrays, que contiene funciones de utilidad para manejar tablas. Dicha clase se encuentra en el paquete java.util.

La mayoría de las funciones están sobrecargadas para soportar tablas de distintos tipos (byte, short, int, long, float, double, etc).

Entre las funciones de la clase Arrays podemos encontrar:

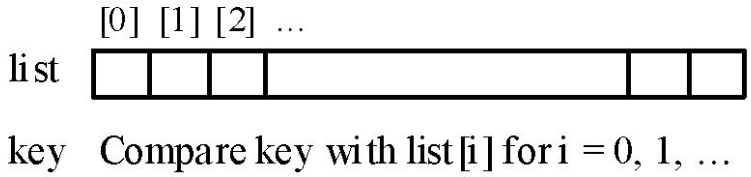
- Buscar un elemento en una tabla ordenada, binarySearch()
- Obtener una subtabla, copyOfRange()
- Comparar si dos tablas son iguales, equals()
- Rellenar una tabla con un valor, fill()
- Ordenar una tabla, sort()
- Mostrar los elementos, toString()

https://docs.oracle.com/javase/8/docs/api/java/util/Arrays.html.

Buscar en Arrays

La búsqueda es el proceso de buscar un elemento específico en una matriz; por ejemplo, descubrir si se incluye un valor determinado en una lista de valores.
La búsqueda es una tarea común en la programación de computadoras. Hay muchos algoritmos y estructuras de datos dedicadas a la búsqueda. En esta sección, se discuten dos enfoques comúnmente utilizados, búsqueda *lineal* y búsqueda *binaria*.

```
public class LinearSearch {
  /** The method for finding a key in the list */
  public static int linearSearch(int[] list, int key) {
    for (int i = 0; i < list.length; i++)
      if (key == list[i])
        return i;
    return -1;
  }
}
```



Búsqueda lineal

El enfoque de búsqueda lineal compara el elemento clave, key, *secuencialmente* con cada elemento en el array list.

El método continúa haciéndolo hasta que la clave coincida con un elemento de list o bien, list se agote sin que se encuentre una coincidencia.

Si se encuentra una coincidencia, la búsqueda lineal devuelve el índice del elemento en el array que coincide con la clave. Si no se encuentra ninguna coincidencia, la búsqueda devuelve -1.

De la idea al desarrollo

```
/** The method for finding a key in the list */
public static int linearSearch(int[] list, int key) {
  for (int i = 0; i < list.length; i++)
    if (key == list[i])
      return i;
  return -1;
}
```

Resultados del método

```
int[] list = {1, 4, 4, 2, 5, -3, 6, 2};
int i = linearSearch(list, 4);  // returns 1
int j = linearSearch(list, -4); // returns -1
int k = linearSearch(list, -3); // returns 5
```

Búsqueda binaria

Para que la búsqueda binaria funcione, los elementos en la matriz ya deben estar ordenados.

Ej., 2 4 7 10 11 45 50 59 60 66 69 70 79

La búsqueda binaria primero compara la clave con el elemento de la mitad de la matriz.

- Si la clave es menor que el elemento medio, solo necesita buscar la clave en la primera mitad de la matriz.
- Si la clave es igual al elemento medio, la búsqueda termina con una coincidencia.
- Si la clave es mayor que el elemento medio, solo necesita buscar la clave en la segunda mitad de la matriz.

De la idea al desarrollo

El método binarySearch devuelve el índice del elemento en la lista que coincide con la clave de búsqueda si está en la lista. De lo contrario, devuelve - 1.

```
public static int binarySearch(int[] list, int key) {
  int low = 0;
  int high = list.length - 1;

  while (high >= low) {
    int mid = (low + high) / 2;
    if (key < list[mid])
      high = mid - 1;
    else if (key == list[mid])
      return mid;
    else
      low = mid + 1;
  }
  return -1 - low;
}
```

El método Arrays.binarySearch

Dado que la búsqueda binaria se utiliza con frecuencia en la programación, Java proporciona varios métodos binarySearch sobrecargados para buscar una clave en una matriz de int, double, char, short, long y float en la clase java.util.Arrays. Por ejemplo, el siguiente código busca las claves en una matriz de números y una matriz de caracteres.

```
int[] list = {2, 4, 7, 10, 11, 45, 50, 59, 60, 66, 69, 70, 79};
System.out.println("Index is " +
  java.util.Arrays.binarySearch(list, 11));

char[] chars = {'a', 'c', 'g', 'x', 'y', 'z'};
System.out.println("Index is " +
  java.util.Arrays.binarySearch(chars, 't'));
```

Devuelve 4

Devuelve –4 (el punto de inserción es 3, así que devuelve -3-1)

Para que funcione el método binarySearch, la matriz debe ordenarse previamente en orden creciente.

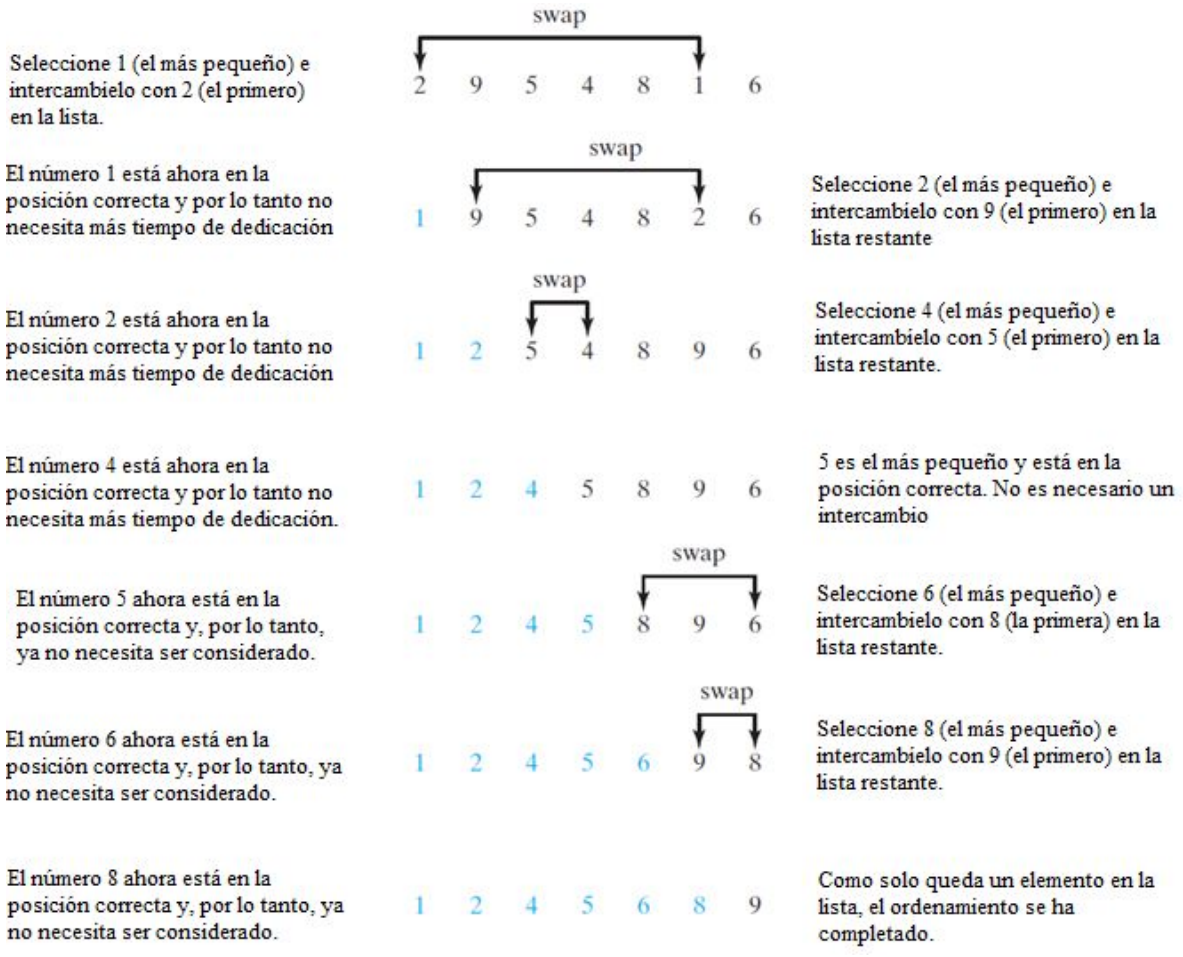
Ordenar matrices

La ordenación, como la búsqueda, también es una tarea común en la programación de computadoras. Se han desarrollado muchos algoritmos diferentes para la clasificación. Esta sección presenta un algoritmo de clasificación simple e intuitivo: *selection sort*.

Ordenación por selección

La ordenación por selección encuentra el número más pequeño en la lista y lo coloca primero. A continuación, encuentra el número más pequeño restante y lo coloca en segundo lugar, y así sucesivamente hasta que la lista contenga un solo número.

Ordenación de matrices por selección



De la idea al desarrollo

```
/** The method for sorting the numbers */
public static void selectionSort(double[] list) {
  for (int i = 0; i < list.length; i++) {
    // Find the minimum in the list[i..list.length-1]
    double currentMin = list[i];
    int currentMinIndex = i;
    for (int j = i + 1; j < list.length; j++) {
      if (currentMin > list[j]) {
        currentMin = list[j];
        currentMinIndex = j;
      }
    }

    // Swap list[i] with list[currentMinIndex] if necessary;
    if (currentMinIndex != i) {
      list[currentMinIndex] = list[i];
      list[i] = currentMin;
    }
  }
}
```

El Método Arrays.sort

Dado que la clasificación se utiliza con frecuencia en la programación, Java proporciona varios métodos de clasificación sobrecargados para ordenar una matriz de int, double, char, short, long y float en la clase java.util.Arrays. Por ejemplo, el siguiente código ordena una matriz de números y una matriz de caracteres.

```
double[] numbers = {6.0, 4.4, 1.9, 2.9, 3.4, 3.5};
java.util.Arrays.sort(numbers);


char[] chars = {'a', 'A', '4', 'F', 'D', 'P'};
java.util.Arrays.sort(chars);
```

Java 8 ahora proporciona Arrays.parallelSort (list) que utiliza el multinúcleo para una clasificación rápida.

El Método Arrays.toString(list)

El método Arrays.toString(list) se puede usar para devolver una representación de cadena para la lista. Esta es una forma rápida y sencilla de mostrar todos los elementos de la matriz.

Por ejemplo, el siguiente código

```
int[] list = {2, 4, 7, 10};
System.out.println(Arrays.toString(list));
```

muestra [2, 4, 7, 10].

Problemas

1.- Escribe un programa que ordene un array de números enteros en orden ascendente usando el método más adecuado.

2.- Modifica el programa anterior para que ordene los números en orden descendente.

3.- Crea un programa que ordene un array de cadenas.

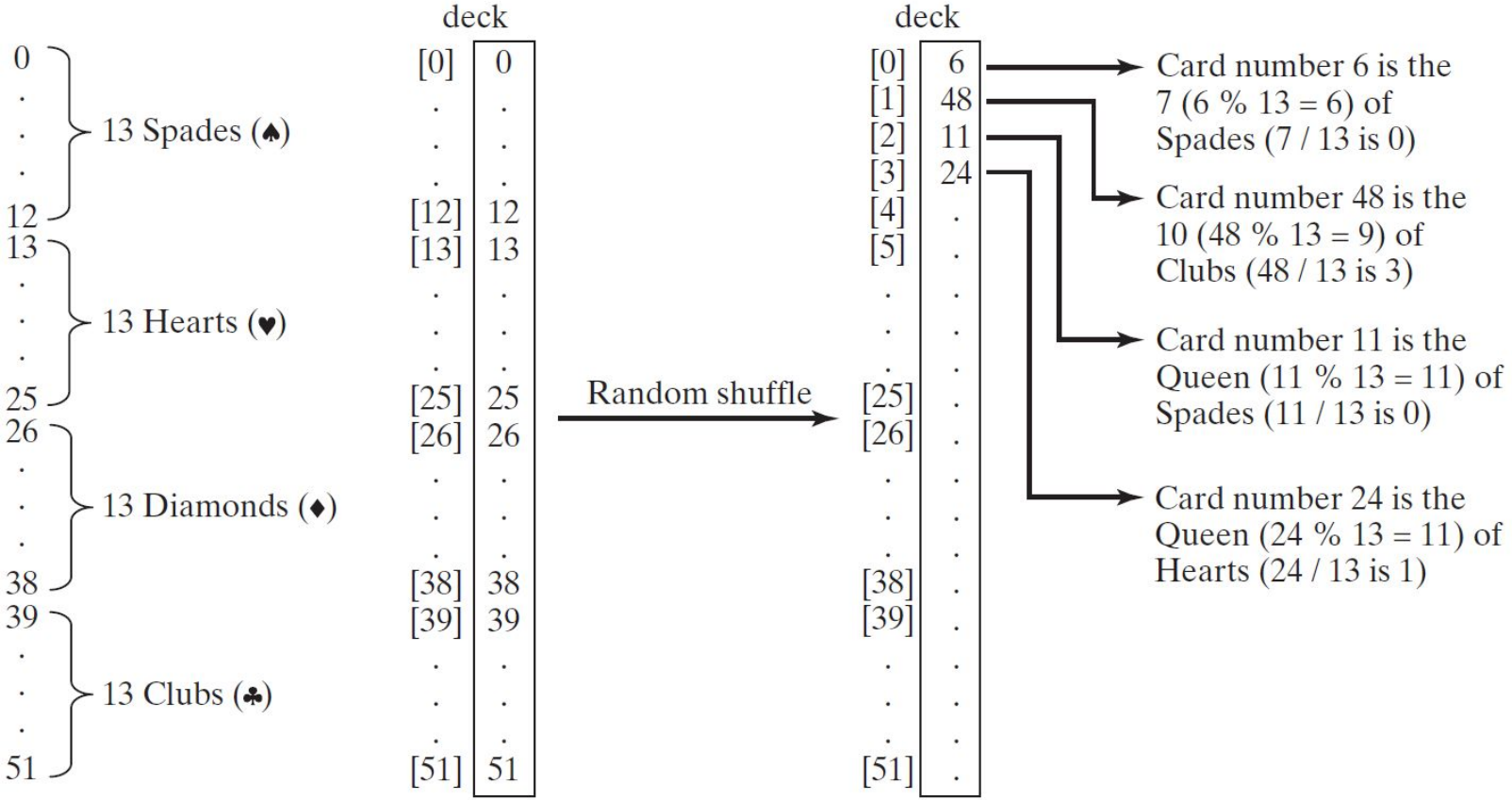
4.- Modifica el programa anterior para que ordene los strings en orden alfabético inverso.

Problema: Baraja de cartas

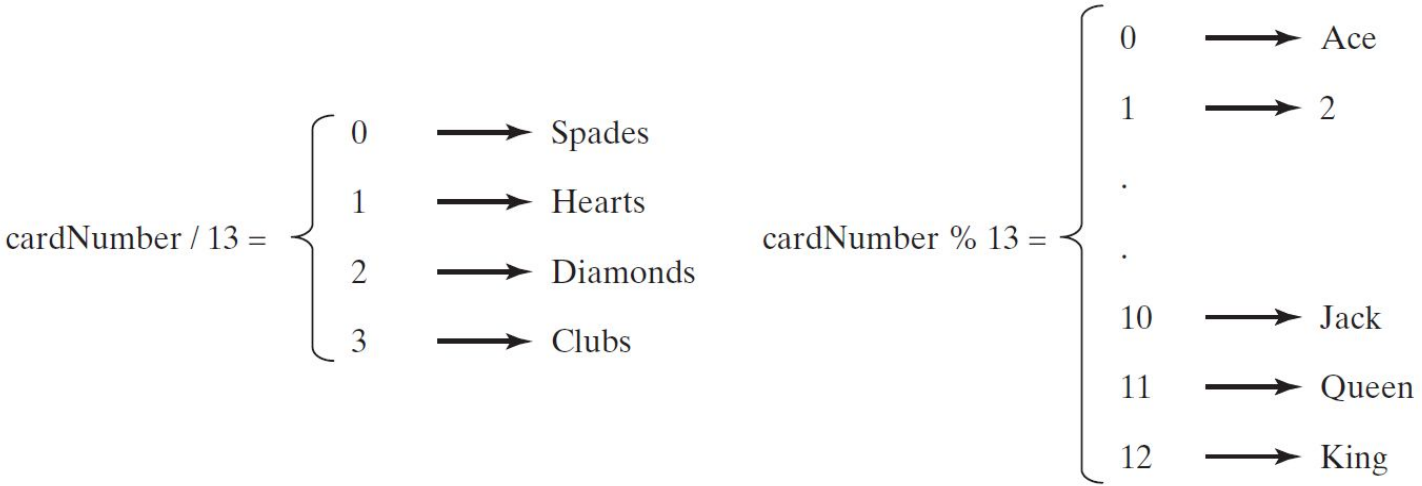
Problema: BARAJA CARTAS. Escribir un programa que elige cuatro cartas al azar de una baraja de 52 cartas. Todas las cartas se pueden representar utilizando una matriz llamada deck (mazo), que contiene los valores iniciales de 0 a 51, de la siguiente manera :

```
int[] deck = new int[52];
// Inizializa cartas
for (int i = 0; i < deck.length; i++)
  deck[i] = i;
```

Problema: Baraja de cartas, cont.



Problema: Baraja de cartas, cont.



Problema: números de la lotería

Supongamos que juegas a la lotería Pick-10.
Cada boleto tiene 10 números únicos que van del 1 al 99. Compras muchos boletos, ya que te gustaría cubrir todos los números del 1 al 99.

Escriba un programa que lea los números del boleto de un archivo y verifique si todos los números están cubiertos. Suponer que el último número en el archivo es 0

Problema: Numeros de la lotería