El futuro digital es de todos

MinTIC

Universidad Pontificia Bolivariana

Vigilada Mineducación

MisiónTIC2022

Arrays y listas

Agrupar objetos

El foco principal de este modulo es introducir algunas maneras en que pueden agruparse los objetos para formar colecciones. En particular, se trata a la clase *ArrayList* como un ejemplo de colecciones de tamaño flexible y al uso de los vectores o arreglos de objetos como colecciones de tamaño fijo. Íntimamente relacionada con las colecciones, aparece la necesidad de recorrer o iterar los elementos que ellas contienen y con este propósito, introducimos tres estructuras de control nuevas: dos versiones del ciclo «*for*» y el ciclo «*while*».

Agrupar objetos en colecciones de tamaño flexible

Cuando escribimos programas, frecuentemente necesitamos agrupar los objetos en colecciones. Por ejemplo:

- Las agendas electrónicas guardan notas sobre citas, reuniones, fechas de cumpleaños, etc.

- Las bibliotecas registran detalles de los libros y revistas que poseen.

- Las universidades mantienen registros de la historia académica de los estudiantes.

Una característica típica de estas situaciones es que el número de elementos almacenados en la colección varía a lo largo del tiempo. Por ejemplo, en una agenda electrónica se agregan nuevas notas para registrar eventos futuros y se borran aquellas notas de eventos pasados en la medida en que ya no son más necesarios; en una biblioteca, el inventario cambia cuando se compran libros nuevos y cuando algunos libros viejos se archivan o se descartan.

Hasta ahora, no hemos hallado en Java ninguna característica que nos permita agrupar un número arbitrario de elementos. Podríamos definir una clase con una gran cantidad de campos individuales, suficiente como para almacenar un número muy grande pero fijo de elementos. Sin embargo, generalmente los programas necesitan una solución más general que la citada. Una solución adecuada sería aquella que no requiera que conozcamos anticipadamente la cantidad de elementos que queremos agrupar o bien, establecer un límite mayor que dicho número

¿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. Para declarar un array se requiere el tipo de dato de los elementos a almacenar y un nombre para el array. Su sintaxis es la siguiente

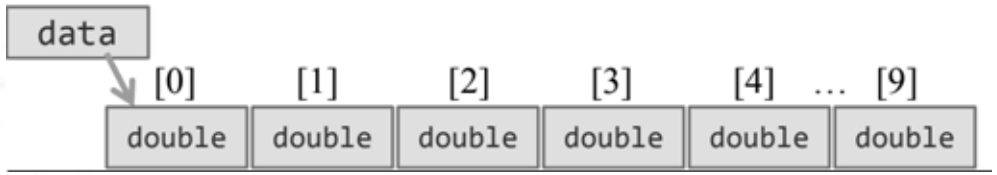
```
double[] data; // declara var. array data
0
double data[];
```

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).

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



Se puede declarar y crear un array al mismo tiempo

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

Declaración 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]	;

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}	;

```
//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"};
```

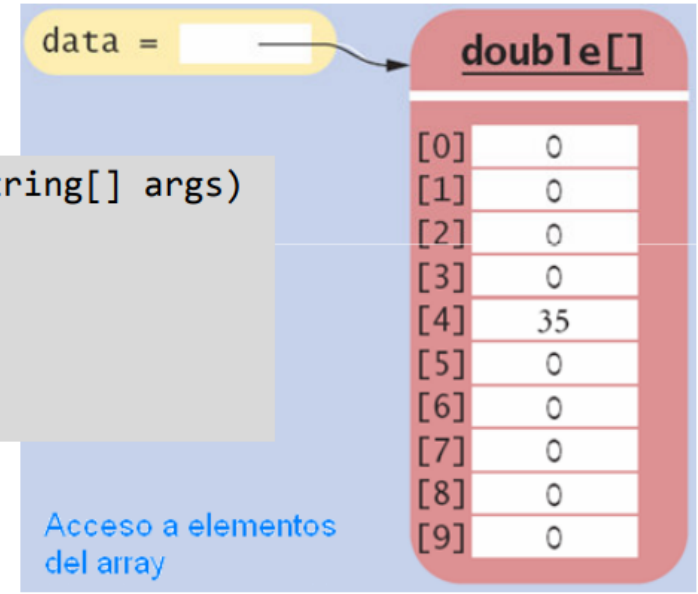
Acceso a los elementos de un array

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.

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



Longitud del array

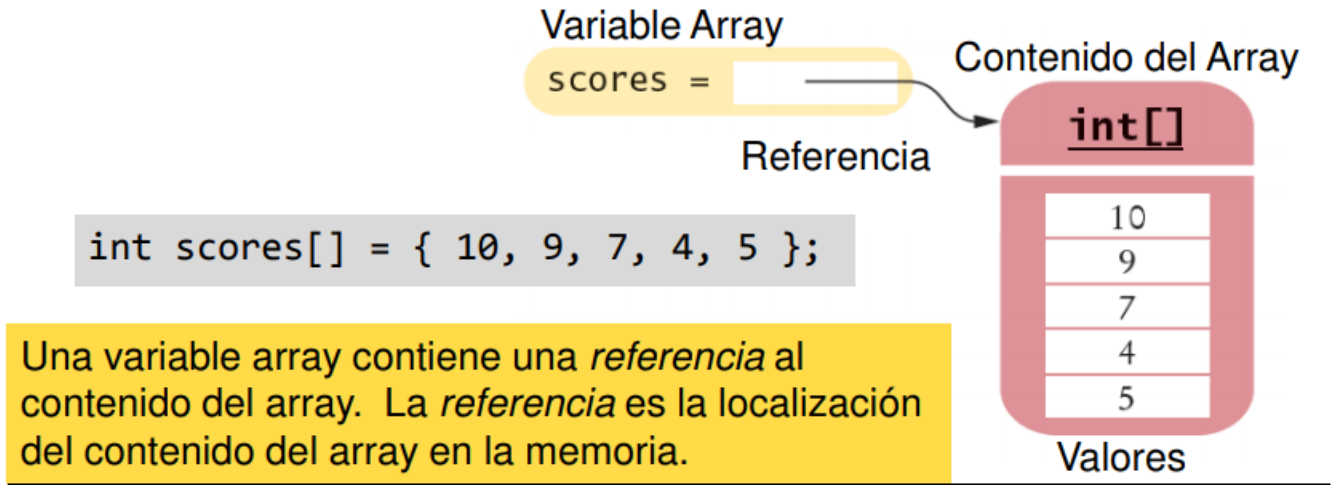
Un array sabe cuántos elementos puede almacenar con *data.length* donde *data* es el nombre del array.

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

Existe una clara diferencia entre Variable array: **El nombre del array** (manejador) y Contenido del array: **Memoria donde se almacenan los valores**

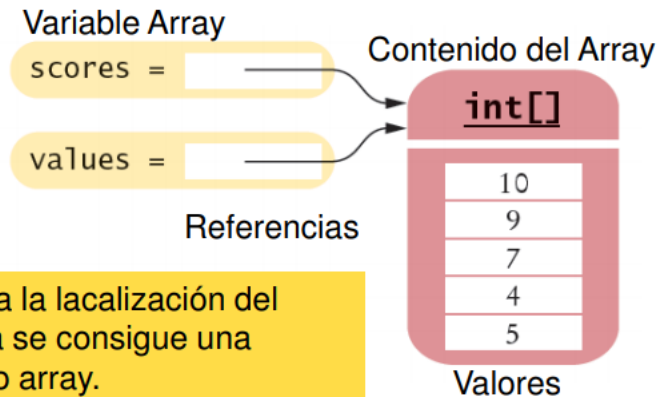



```
int scores[] = { 10, 9, 7, 4, 5 };
```

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

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.

Uno de los errores mas frecuentes es olvidarse de asignar memoria para el contenido del array, por lo que el programa arroja un 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
```

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}};
```

Declaración de Arrays multidimensionales

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

Ejemplo de arreglos en Java

Como los arreglos en Java son objetos, éstos tienen atributos que proporcionan información del objeto. El atributo length regresa como resultado el tamaño del arreglo.

Ejemplo 1. Declarar un arreglo de enteros de tamaño 30 y poner en cada uno de sus elementos su propio índice multiplicado por dos. Imprimir cada uno de los elementos del arreglo.

```
public class Main {

  public static void main( String[] args ) {

    Integer[] arregloDeEnteros; // declaracion del arreglo

    arregloDeEnteros = new Integer[30]; // instancia del arreglo

    for ( int i = 0; i < arregloDeEnteros.length; i++ ) {
      arregloDeEnteros[i] = 2 * i;
      System.out.println( "A[" + i
                          + "]= " + arregloDeEnteros[i] + "\n");
    }
  }
}
```

Arreglos como parámetros de entrada a un método

Los arreglos pueden pasarse como parámetros de entrada a un método. Por ejemplo, si tenemos el arreglo:

```
Double[] temperaturas = new Double[5];
```

El llamado a un método que recibe un arreglo como parámetro se hace de la siguiente manera:

```
despliegaArreglo( temperaturas );
```

Es decir, basta con enviar como parámetro actual el nombre del arreglo, el cual es una referencia al inicio del arreglo. El método que recibe el arreglo debe tener declarado el arreglo como parámetro formal con la siguiente sintaxis:

```
public tipo nombreMetodo( tipoElemento[] nombreArreglo)
```

En el caso de nuestro ejemplo, tendríamos:

```
public void despliegaArreglo( Double[] A ) {
  for ( int i = 0; i < A.length; i++ ) {
    System.out.println( "temperatura " + i + " =" + A[i] + " ");
  }
}
```

El hecho de que el método reciba una referencia al arreglo, implica que los cambios que el método lleve a cabo los hace directamente sobre el arreglo original. Por lo tanto, cuando se envía un arreglo a un método no es necesario regresarlo como valor de retorno. Sin embargo, puede haber casos en los que sea necesario regresar un arreglo como valor de retorno desde un método.

Arreglos como valor de retorno de un método

También es posible que un método regrese un arreglo, en este caso la sintaxis para el método es la siguiente:

```
public tipoElemento[] nombreMetodo( parametros )
```

Cuando se invoca a un método que regresa un arreglo, se debe hacer la asignación en una variable de tipo arreglo, por ejemplo, en la variable *datosTemperatura*.

Supongamos que se preguntó previamente por el valor de *tamanio*.

```
Double[] datosTemperatura = new Double[ tamanio ];
```

Entonces, la llamada a un método que captura los datos del arreglo y regresa el arreglo capturado se hace de la siguiente forma:

```
datosTemperatura = capturaArreglo( tamanio );
```

A continuación se presenta el programa completo, que también despliega los valores del arreglo.

```
public class ArregloMain {

  public static void main( String[] args ) {
    Integer tamanio;

    tamanio = LectorDeTeclado.capturarEntero(
                              "De que tamanio es el arreglo?")
    Double[] datosTemperatura = new Double[ tamanio ];

    datosTemperatura = capturarArreglo( tamanio );
    desplegarArreglo( datosTemperatura );
  }

  // Método para capturar los valores de un arreglo
  public static Double[] capturarArreglo( int tamanio ) {

    Double[] datos = new Double[ tamanio ];

    for ( int i = 0; i < tamanio; i++ ) {
      datos[i] = LectorDeTeclado.capturarEntero(
                              "Dato " + i + " ? ")
    }

    return datos; // regresa la referencia al arreglo
  }

  // Método para desplegar los valores de un arreglo
  public static void despliegaArreglo( Double[] A ) {

    for ( int i = 0; i < A.length; i++ )
      System.out.println( "temperatura " + i + " =" + A[i] + " ");
  }
}
```

Ejercicios con arreglos

Ejercicio 1:

Hacer las siguientes declaraciones:

1. Declarar e instanciar un arreglo de 17 *Integer* llamado A.
2. Declarar e instanciar un arreglo de 1150 *Double* llamado B.
3. Declarar e instanciar un arreglo de 300 *Float* llamado C.
4. Declarar e instanciar un arreglo de 3 elementos de la clase *Manecilla* llamado *manecillas.*

Solución:

```
Integer[] A = new Integer[ 17 ];
Double[] B = new Double[ 1150 ];
Float[] C = new Float[ 300 ];
Manecilla[] manecillas = new Manecilla[ 3 ];
```

Ejercicios con arreglos

Ejercicio 2:

Suponiendo que ya tenemos un arreglo de int llamado A, de tamaño n, ¿cómo haces el código que pide al usuario cada uno de sus elementos?

Solución:

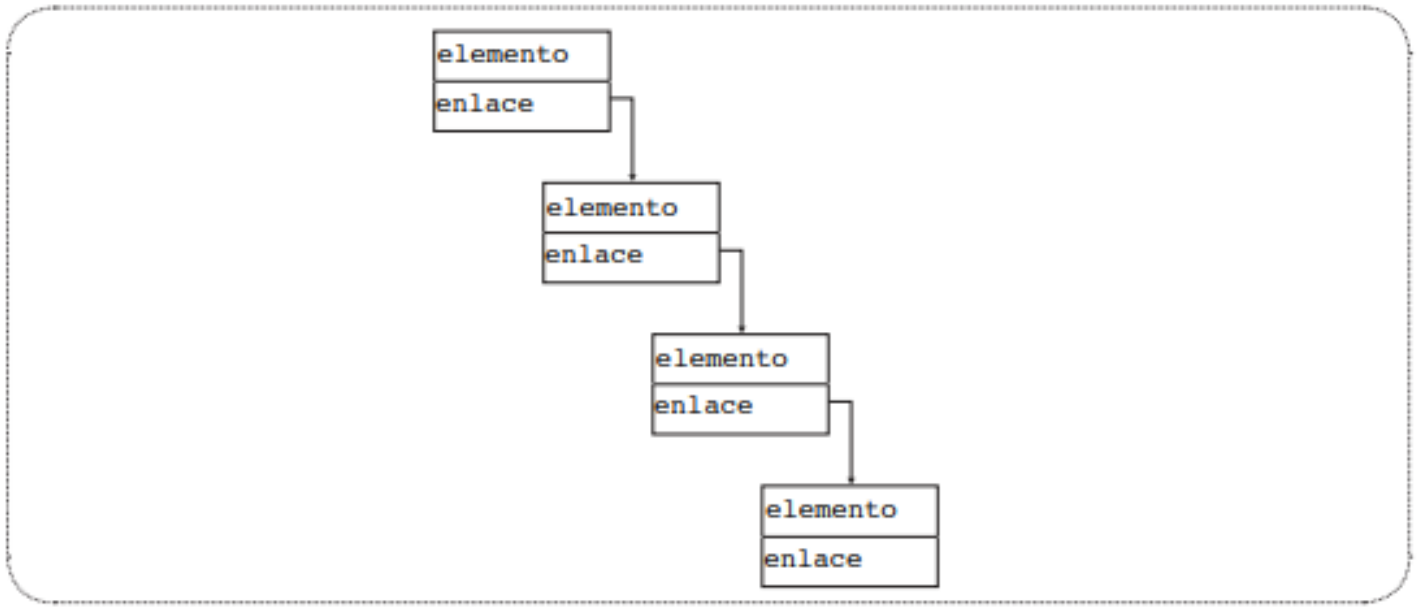
```
Scanner input = new Scanner( System.in );

for ( int i = 0; i < n; i++) {
  System.out.println( "A[" + i + "]=?" );
  A[i] = input.nextInt();
}
```

Listas con objetos

Concepto de lista ligada

Las listas ligadas se forman con “cajas” que contienen un elemento y un enlace a la siguiente caja



Ya existe en Java una clase que se llama *LinkedList*. La reutilización de código permite trabajar con listas ligadas sin tener que programar todo su funcionamiento. Para poder hacer uso de la clase *LinkedList* es necesario importarla.

```
import java.util.LinkedList;
```

El elemento que contiene la lista ligada es un objeto de cualquier clase, incluyendo las *wrapper* (*Integer, String, Double,* …). La sintaxis para declarar e instanciar una lista ligada de una *ClaseX*, es la siguiente:

```
LinkedList<ClaseX> nombreLista;
nombreLista = new LinkedList<ClaseX>();
```

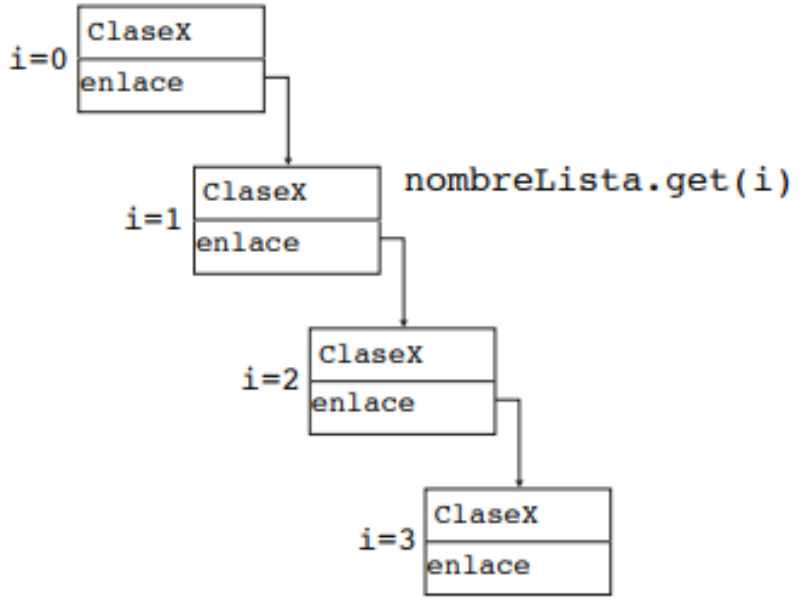
O todo en un solo renglón:

```
LinkedList<ClaseX> nombreLista = new LinkedList<ClaseX>();
```

Al indicar entre los signos <> la clase a utilizar, estamos estableciendo que esta lista ligada contendrá exclusivamente elementos de ese tipo. De esta forma, si queremos insertar elementos que no correspondan al tipo especificado, el compilador de Java lo detectará y nos enviará un mensaje de error. Lo mismo sucede si lo que queremos es recuperar un elemento de la lista y hacer la referencia a él con un tipo que no corresponde.

Esta clase tiene algunos métodos de mucha utilidad, por ejemplo, podemos añadir un elemento al principio de la lista ligada con el método addFirst() o podemos añadir un elemento al final con el método addLast().

Para obtener un elemento i de la lista ligada usamos el método get(i)

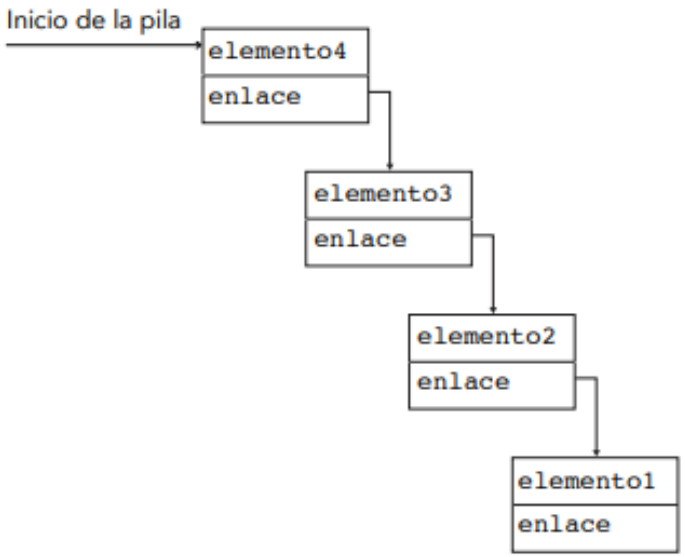


Concepto de pila

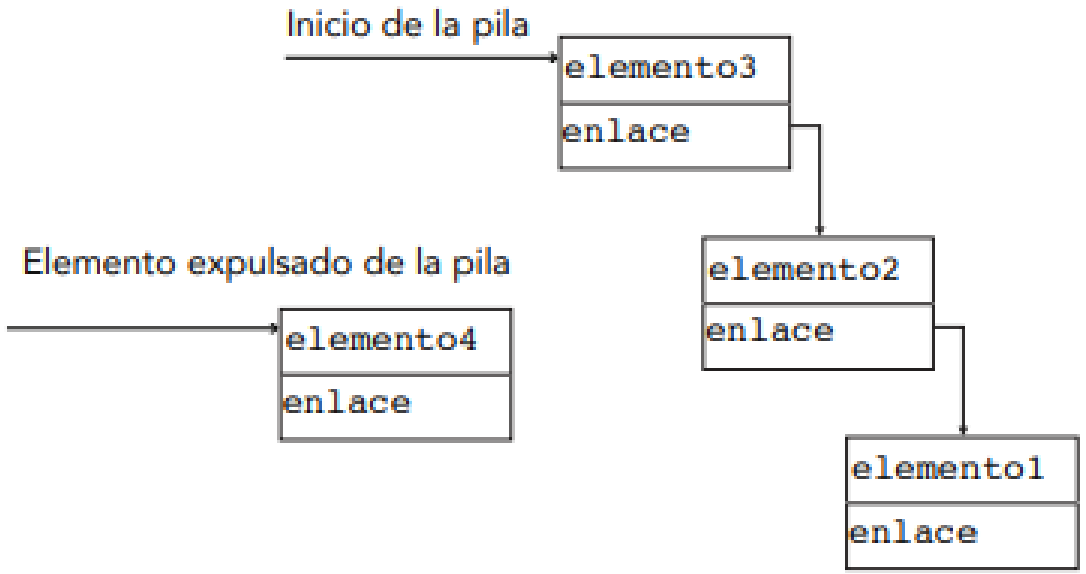
Concepto de pila. En una pila se almacenan elementos uno sobre otro, de tal forma que el primer elemento disponible es el último que fue almacenado. A una pila también se le conoce como lista LIFO (Last In First Out), es decir, el primero en entrar es el último en salir. Una pila en la programación puede compararse con una pila de platos o de tortillas. Cada plato (o tortilla) nuevo se coloca hasta arriba de la pila. De esta forma, si deseamos quitar un plato, el primero disponible resulta ser el último que se guardó en la pila de platos.

Existen dos operaciones básicas para trabajar con pilas, que son *push()* y *pop()*. La operación *push(),* que en inglés significa empujar, se encarga de poner hasta arriba de la pila un elemento nuevo. En la figura se ilustra una pila en la que se hicieron cuatro operaciones push() en el siguiente orden:

- push(elemento1)
- push(elemento2)
- push(elemento3)
- push(elemento4)



La operación *pop()* en este contexto puede traducirse como expulsar o sacar de la pila. *pop()* entrega un elemento expulsado de la pila y el inicio de la pila apunta ahora al siguiente elemento. Como se ilustra en la figura, el último elemento que entró (elemento4) fue el primero en salir



Ejercicio de pilas

Hacer un programa que inserte 4 enteros a una lista ligada. Posteriormente, debe sacar cada uno de los elementos de la lista y guardarlos en una pila.

Finalmente, debe sacar los elementos de la pila e imprimirlos

```
import java.util.*;

public class PilaMain {

  public static void main( String[] args ) {

    Stack<Integer> pila = new Stack<Integer>();
    LinkedList<Integer> lista = new LinkedList<Integer>();

    lista.addFirst( 1 );
    lista.addFirst( 2 );
    lista.addFirst( 3 );
    lista.addFirst( 4 );

    Integer elemento;
    for( int i = 0; i < 4; i++ ) {
      elemento =  lista.get( i );
      pila.push( elemento );
    }

    System.out.println( "elementos de la pila: " );
    for( int j = 1; j <= 4; j++ ) {
      elemento = ( Integer ) pila.pop();
      System.out.println( "elemento " + j + " = " + elemento );
    }
  }
}
```

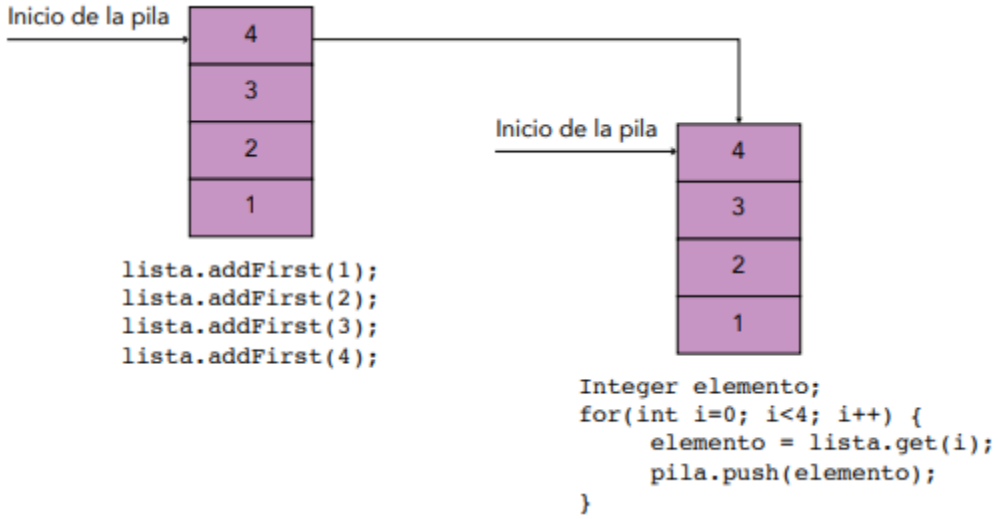
Saca un elemento de la lista y lo guarda en la pila.

Saca un elemento de la pila y lo muestra en pantalla.

Esta es la salida del programa anterior

```
elementos de la pila:
elemento 1 = 1
elemento 2 = 2
elemento 3 = 3
elemento 4 = 4
```

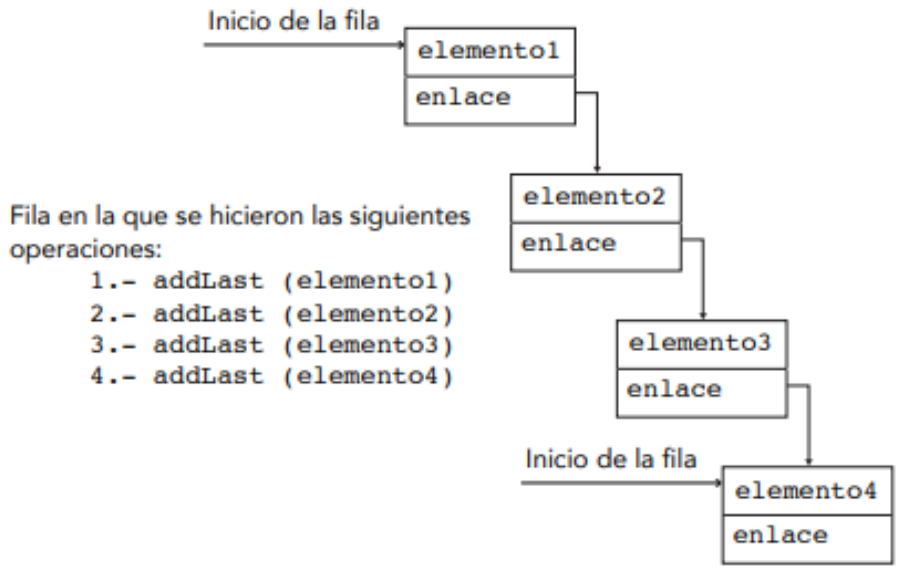
Para entender mejor este resultado, miremos en la figura la estructura de la lista ligada y de la pila. Como los números se añaden al principio de la lista, estos quedan en orden inverso, al ir sacando los elementos de la lista comenzando por el principio, el orden se vuelve a invertir.



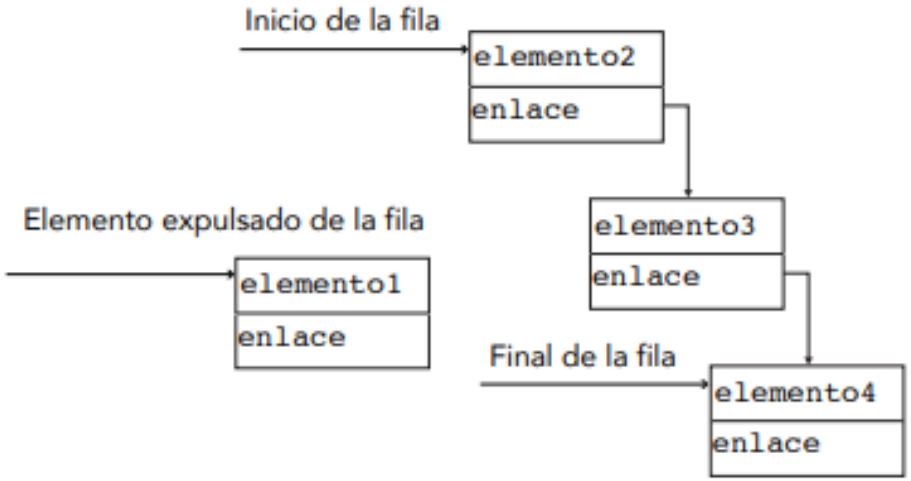
Concepto de fila (cola)

En una *fila* o *cola* el primer elemento que se saca de la lista es el primero que entró. A una cola también se le conoce como lista First In First Out (FIFO), es decir, el primero en entrar es el primero en salir. Una fila o cola en la programación funciona como una fila del banco o del supermercado.

Existen dos operaciones básicas para trabajar con filas, que son *formar()* y *despachar()*. La operación *formar()* se encarga de poner al final de la fila un elemento nuevo. El final de la fila apuntará ahora hacia el nuevo elemento que se formó. Podemos utilizar la clase *LinkedList* para trabajar con filas (colas). La operación formar se efectúa con el método *addLast()* de esta clase, el cual agrega el elemento al final de la fila, como se muestra en la figura.



La otra operación básica cuando se trabaja con filas es la función *despachar(),* que saca de la fila al primer elemento. En la figura IV-8 se ilustra la operación *despachar(),* que entrega el primer elemento y así el incio apunta al siguiente elemento de la fila.



La operación *despachar()* se puede efectuar con el método *removeFirst()* de la clase *LinkedList*, el cual recupera y elimina de la fila el primer elemento.

Si la fila está vacía, lanza la excepción *NoSuchElementException.*

Ejercicio de filas (colas)

El siguiente programa inserta 4 enteros a una fila, posteriormente saca cada uno de los elementos de la lista y los guarda en una fila. Finalmente, saca los elementos de la fila y los va imprimiendo:

```
import java.util.LinkedList;

public class FilaMain {

  public static void main( String[] args ) {

    LinkedList<Integer> fila = new LinkedList<Integer>();
    Integer elemento;

    fila.addLast( 1 );
    fila.addLast( 2 );
    fila.addLast( 3 );
    fila.addLast( 4 );

    while ( !fila.isEmpty() ) {
      elemento = fila.removeFirst();
       System.out.println( "Se atendio al elemento: " + elemento
                          + " de la fila");
    }

    System.out.println( "La fila esta vacia" );
  }
}
```

El método `isEmpty()` regresa `true` cuando ya no hay elementos en la lista.

Esta es la salida del programa anterior

```
Se atendió al elemento:  1 de la fila
Se atendió al elemento:  2 de la fila
Se atendió al elemento:  3 de la fila
Se atendió al elemento:  4 de la fila
La fila está vacía
```

Otra manera de implementar lo mismo que en el ejemplo anterior es utilizando el método *pollFirst()*, el cual recupera y elimina de la fila el primer elemento. Además, si la fila está vacía regresa *null.*