



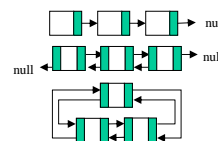
# Representación de datos

## Estructuras de Datos: Arrays y Listas

1

## Introducción

- ¿Qué **son**?
- ¿Por qué son **necesarias**?
- Estructuras simples
  - **Arrays** 
    - Declaración, creación e inicialización
    - Inserción, eliminación y localización de elementos
    - Arrays multidimensionales
  - **Listas estáticas** 
    - Declaración, creación e inicialización
    - Inserción, eliminación y localización de elementos
  - **Listas dinámicas**
    - Listas simplemente enlazadas
    - Listas doblemente enlazadas
    - Listas circulares



2

## Estructuras de datos

### ¿Qué son?

Una **estructura de datos** consiste en:

- Una **representación** de los datos
- Conjunto de **operaciones** permitidas sobre esos datos

- Las **operaciones** más frecuentes sobre los datos son:
  - Inserción
  - Eliminación
  - Búsquedade algún dato dentro de la estructura
- Las **principales diferencias** entre las distintas estructuras son:
  - la forma de **organizar** los datos de manera útil para un acceso eficiente, y
  - las **restricciones** que imponen sobre las operaciones (ej. acceso sólo al primer o al último elemento introducido)

3

## Estructuras de datos

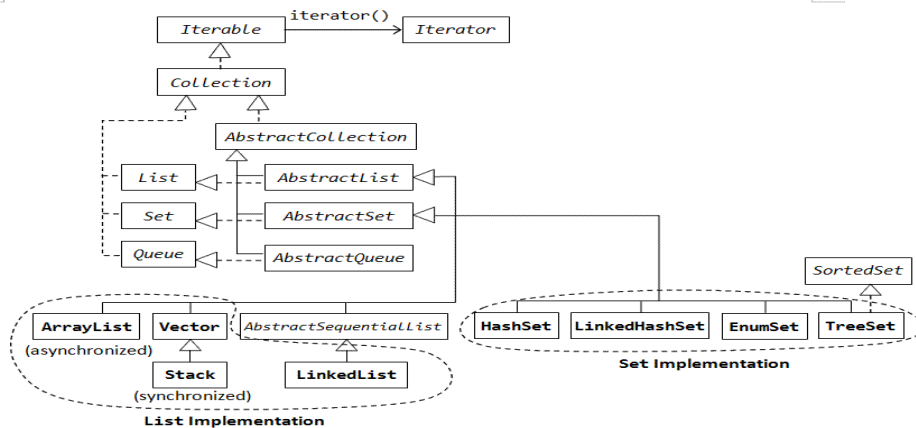
### ¿Por qué son útiles?

- **Reutilización**: Constituyen una forma de almacenar datos que pueden ser reutilizadas
- **Abstracción**: Separan la declaración de la funcionalidad (interfaz) de la implementación concreta
  - Esto permite modificar la estructura de datos sin modificar los programas que las utilizan
    - Definiendo variables y parámetros de tipo interface
    - Instanciando y pasando objetos de la clase concreta
- **Almacén de Referencias**: Estas estructuras almacenan referencias y no copias internas de los datos.

4

## Estructuras de datos predefinidas

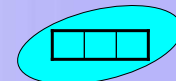
- Colecciones de tamaño predeterminado: p.ej **array**
- Colecciones de tamaño indefinido: **List**, **Stack**, **Queue**, **Tree**, **Set**



[http://www3.ntu.edu.sg/home/ehchua/programming/java/J5c\\_Collection.html](http://www3.ntu.edu.sg/home/ehchua/programming/java/J5c_Collection.html)

## Arrays (Matrices)

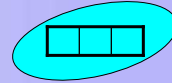
¿Qué son?



- Colección de entidades del **mismo tipo** almacenadas en una unidad
- Los tipos pueden ser tanto **primitivos** como **referencias a objetos**
- El **acceso** a cada elemento del array se realiza mediante el operador de **indexación [ ]**
- El atributo **length** nos permite saber el n° de elementos que contiene el array

## Arrays (Matrices)

### Declaración, Creación, Inicialización



- **Declaración:** Consiste en asignar un **identificador** al array y decir de qué **tipo** són los elementos que va a almacenar.

- Se puede hacer de 2 formas
- Después de la declaración **aún no se ha asignado memoria** para almacenar el array
- No podemos acceder a su contenido

```
tipo nombreArray[];  
tipo[] nombreArray;
```

#### Valores por defecto:

```
int, short, long = 0  
float, double = 0.0  
booleanos = false  
String = null  
Object = null
```

- **Creación:** Consiste en reservar espacio en memoria para el array
- Es necesario utilizar **new** y especificar

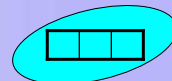
**tamaño** del array `nombreArray = new tipo[numPosiciones];`

- Una vez creado el array sus elementos tienen **valores por defecto** hasta que el array sea inicializado
- Los elementos están almacenados en **posiciones contiguas** de la memoria

7

## Arrays (Matrices)

### Declaración, Creación, Inicialización



- **Inicialización:** Consiste en **dar valores** a los distintos elementos del array podemos hacerlo de varias formas:

- Elemento a elemento

```
nombreArray[0] = elemento0;  
nombreArray[1] = elemento1;  
...
```

- Mediante un bucle

```
for(int i = 0; i < nombreArray.length; i++){  
    nombreArray[i] = elemento - i;  
}
```

- Por asignación directa

```
tipo[] nombreArray = {elem1, elem2, elem3, ...};
```

8

## Ejemplos: Arrays (Matrices)

### Declaración, Creación, Inicialización



#### Arrays de tipos básicos

```
int[] a;           //Declara
a = new int[3]     //Crea
a[0]=1;           //Inicializa
a[1]=2;
a[2]=3;
```

```
int[] a= new int[3] //Declara y Crea
a[0]=1;             //Inicialización
a[1]=2;
a[2]=3;
```

```
int[] a= new int[3] //Declara y crea
for(int i=0; i<a.length;i++){ //Inicializa
    a[i]=i+1;
}
```

```
int a[] = {1, 2, 3}; //Declaración, creación, inicialización
```

#### Arrays de objetos (Tipos de referencia)

```
MiClase[] a;       //Declara
a = new MiClase[3] //Crea
a[0]=new MiClase(param1);
a[1]=new MiClase(param2);
a[2]=new MiClase(param3);
```

```
MiClase[] a= new MiClase[3]
//inicializa
a[0]=new MiClase(param1);
a[1]=new MiClase(param2);
a[2]=new MiClase(param3);
```

```
MiClase[] a= new MiClase[3]
//inicializa
for(int i=0; i<a.length;i++){
    a[i]=new MiClase(param-i);
}
```

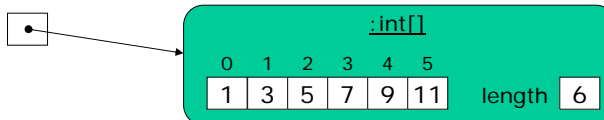
```
MiClase[] a = {new MiClase(param1), new MiClase(param2), new MiClase(param3)};
```

## Arrays (Matrices)

### tipo primitivo vs tipo referencia

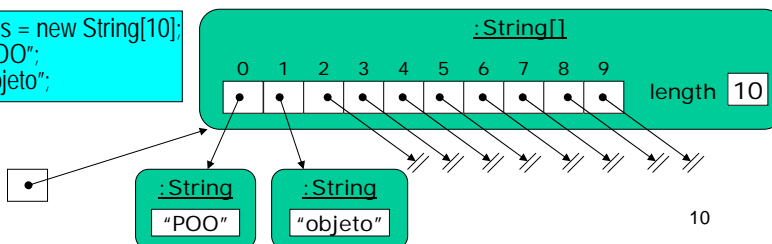
```
int[] numbers = new int[6];
for(int i = 0; i < numbers.length;i++)
    numbers[i] = 2*i + 1;
```

numbers



```
String[] notes = new String[10];
notes[0] = "POO";
notes[1] = "objeto";
```

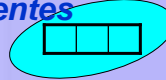
notes



10

## Arrays (Matrices): Errores frecuentes

### Declaración, creación, inicialización



```
public class EjemplosMatrices{
    public static void main(String[] args){
        double[] miMatriz;
        System.out.println(miMatriz[0]);
    }
}
```

**MAL**

compilar

Falla la compilación

variable miMatriz may not have been initialized



Cuando la matriz sólo ha sido **declarada** no podemos acceder a sus elementos. El programa no compilaría y daría un **error**

## Arrays (Matrices) Errores frecuentes

### Declaración, creación, inicialización



```
public class EjemplosMatrices2{
    public static void main(String[] args){
        int[] miMatrizDeEnteros= new int[10];
        float[] miMatrizDeReales[]= new float[10];
        boolean[] miMatrizDeBooleanos= new boolean[10];
        char[] miMatrizDeCaracteres= new char[10];
        String[] miMatrizDeStrings= new String[10];
        Object[] miMatrizDeObjetos= new Object[10];
        System.out.println("Entero por defecto: " + miMatrizDeEnteros[0]);
        System.out.println("Real por defecto: " + miMatrizDeReales[0]);
        System.out.println("Booleano por defecto: " + miMatrizDeBooleanos[0]);
        System.out.println("Carácter por defecto: " + miMatrizDeCaracteres[0]);
        System.out.println("String por defecto: " + miMatrizDeStrings[0]);
        System.out.println("Objeto por defecto: " + miMatrizDeObjetos[0]);
    }
}
```

compilar

Ejecutar



Cuando la matriz sólo ha sido **declarada y creada** pero **no inicializada** podemos acceder a sus elementos pero estos tienen su **valor por defecto**

Entero por defecto: 0  
Real por defecto: 0.0  
Booleano por defecto: false  
Carácter por defecto:  
String por defecto: null  
Objeto por defecto: null

## Arrays

### Iteración

- Habitualmente queremos repetir varias veces un conjunto de acciones
  - Por ejemplo, imprimir las notas de la agenda una a una
- La mayoría de los lenguajes de programación incluyen sentencias de bucle para hacer esto:
  - `while`, `for`, `repeat`, ...
- Los bucles permiten controlar cuántas veces se repite un conjunto de acciones
  - En el caso de las colecciones es habitual repetir acciones para cada objeto de la colección particular

## Arrays

### Iteración: el bucle `for`

- Hay dos variaciones del bucle `for`: `for` y `for-each`.
- El bucle `for` se utiliza normalmente para iterar un número fijo de veces
  - Generalmente se utiliza con una variable que se incrementa (o decrementa) una cantidad fija en cada iteración
  - Es parecido al `while`
- El bucle `for-each` repite el cuerpo del bucle para cada objeto de una colección (patrón `iterable`)
  - Más fácil de escribir
  - Más seguro: hay garantía de que acabará

14

## Arrays

Bucle **for** vs. **while** vs. **for-each**: ejemplo `printNotes()`

- **for**

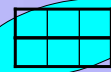
```
String[] notes;

for(int i = 0; i < notes.length; i++)
    System.out.println(notes[i] + ",");
```
  - **while**

```
int i = 0;
while(i < notes.length){
    System.out.println(notes[i] + ",");
    i++;
}
```
  - **for-each**

```
for(String note: notes)
    System.out.println(note + ",");
```
- Mientras el valor del índice *i* es menor que el tamaño, imprimir la siguiente nota, e incrementar el índice
- Para cada nota en la colección *notes*, imprimir la nota

## Arrays Multidimensionales



- Es un array en el que el acceso a los elementos se hace utilizando más de un índice

	0	1	2
0	A	B	C
1	D	E	F
2	G	H	I

`A[0][2]='C'`

```
char[][] a;      //Declara
a = new char[3][3] //Crea
a[0][0]='A';     //Inicializa
...
```

		0	1	2
0	0	a	b	c
1	0	d	e	f
2	0	g	h	i

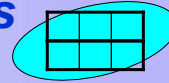
`A[0][2][1]='I'`

```
char[][][] a;    //Declara
a = new char[3][3][3] //Crea
a[0][0][0]='g';   //Inicializa
...
```



## Arrays Multidimensionales

### Ejemplos



#### Declarar y crear directamente

```
//Declaración y Creación
String[ ][ ] miMatriz = new String[3][4]
```

null	null	null	null
null	null	null	null
null	null	null	null

#### Declarar y crear por pasos

```
int[ ][ ] miMatriz ;           // Declarar el array
miMatriz = new int[numFilas][numCols]; // Crear el array de referencia
```

#### Otros ejemplos

```
// Matriz 3x3 inicializada a 0
int[ ][ ] a= new int[3][3];
```

0	0	0
0	0	0
0	0	0

```
int[ ][ ] b= {{1, 2, 3},
               {4, 5, 6}};
```

1	2	3
4	5	6

## Estructuras de datos

### the Collection API

- El interfaz `Collection<T>` declara la funcionalidad básica que debe tener cualquier colección de datos
- Facilita el uso o procesamiento de cualquier estructura de datos, *independiente* de su implementación concreta
- Operaciones:
  - int `size()`
  - boolean `isEmpty()`
  - boolean `contains(Object o)`
  - `Iterator<T> iterator()`
  - `Object[] toArray()`
  - boolean `add(T element)`
  - boolean `remove(Object o)`
  - void `clear()`



## Estructuras de datos

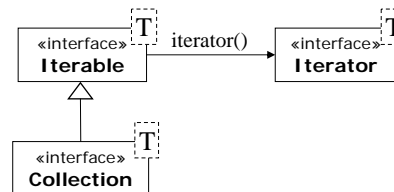
### Iteradores

```
public interface Iterable<T> {
    public Iterator<T> iterator();
}
```

```
public interface Collection<T> implements Iterable<T>
```

- Se puede iterar sobre cualquier colección
- Se puede utilizar el bucle “for-each” para cualquier colección

```
public interface Iterator<T> {
    public boolean hasNext();
    public T next();
    public void remove();
}
```



19

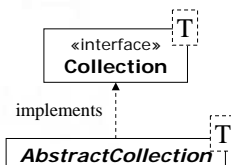
## Estructuras de datos

### the Collection API

- La clase `AbstractCollection<T>` implementa la funcionalidad básica, que es independiente de la implementación concreta
  - Algunos métodos se implementan en base a otros métodos
  - Mediante la herencia y poliformismo, en tiempo de ejecución, se ejecutarán los métodos (i.e. `size()`) de la clase concreta

```
/** Test if this collection es empty */
public boolean isEmpty() {
    return size() == 0;
}
```

- Los métodos restantes son abstractos o producen una excepción (`java.lang.UnsupportedOperationException`)
- Se implementarán en la clase concreta



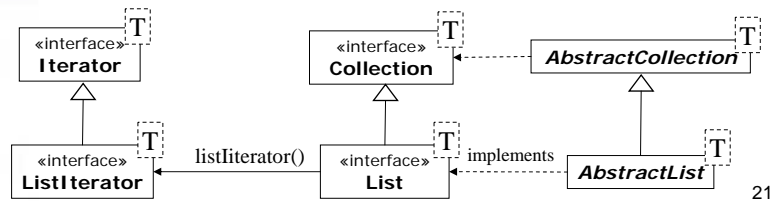
```
/** Adds element to this collections */
public boolean add(T element) {
    throw new UnsupportedOperationException();
}
```

20

## Listas (List)

### ¿Qué son?

- Define el concepto de **secuencia** sobre la colección de datos, es decir, **el orden es importante**
- El interfaz **List<T>** declara la funcionalidad básica que debe tener cualquier lista de datos
- Declara las operaciones **en base a una posición**:
  - void **add**(int index, T element)
  - T **remove**(int index)
  - T **get**(int index) y T **set**(int index, T element)
  - int **indexOf**(Object obj) y int **lastIndexOf**(Object obj)
  - ListIterator<T> **listIterator**(int index)

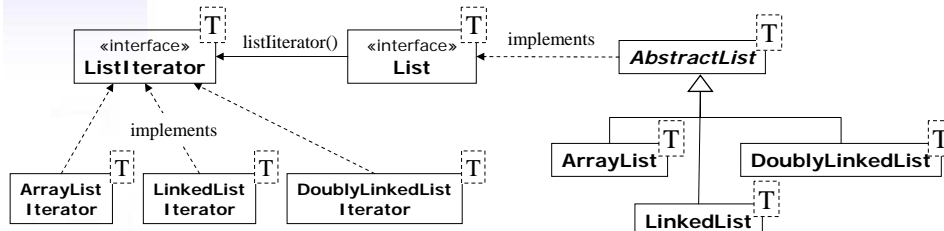


21

## Listas (List)

### Clasificación y clases concretas

- Según organización en memoria:
  - **Estática**: ej. ArrayList
  - **Dinámica**: **Simplemente** vs. **Doblemente** enlazada
    - ej. LinkedList, DoublyLinkedList
- Según recorrido: **Lineal** vs. **Circular**
  - ej. LinkedList vs. CircularlyList



22

**OJO:** estas clases concretas no son las estandares de Java

## ArrayList

### ¿Qué son?



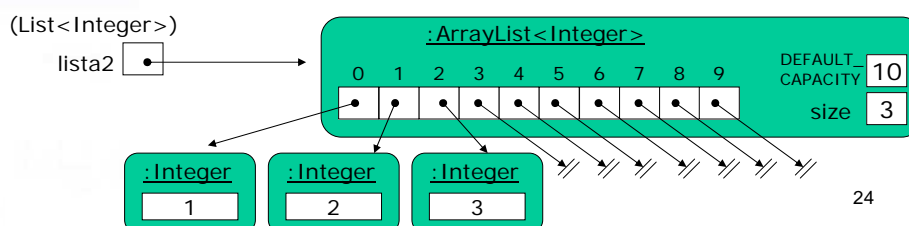
- Colección de elementos (referencias a objetos) de cualquier **tipo** (clase)
- Su **implementación** se basa en un **array**
- Puede crecer o reducirse **dinámicamente** según el nº de elementos
- Permite acceder a los elementos mediante un **índice** pero **no mediante [ ]**, si no utilizando los métodos **get**(int index) o **set**(int index, T element)
- Tamaño:
  - La variable **DEFAULT\_CAPACITY** mantiene el nº de elementos que puede contener la lista inicialmente
  - El método **size()** devuelve los que en realidad contiene
  - Cuando necesite crecer **duplica la capacidad** actual

## ArrayList

### Métodos constructores

Constructores	Significado
<code>ArrayList()</code>	Crea una lista vacía
<code>ArrayList(Collection &lt;? extends T&gt; other)</code>	Crea una lista copiando los valores de la colección other

```
List<Integer> listaVacia = new ArrayList<Integer>;
List<Integer> lista = Arrays.asList(1, 2, 3); //Crea una lista, pero no sabemos de que clase concreta
List<Integer> lista2 = new ArrayList<Integer>(lista);
```



## ArrayList



### Métodos de acceso

Métodos acceso	Significado
boolean <b>contains</b> (Object elem)	Comprueba si la lista contiene el objeto <i>elem</i>
boolean <b>containsAll</b> (Collection<? extends T> col)	Comprueba si la lista contiene todos los objetos de la colección <i>col</i>
Iterator<T> <b>iterator</b> ()	Devuelve un iterador para recorrer la lista desde el inicio
Object[] <b>toArray</b> ()	Devuelve un array con las referencias a los objetos de la colección
T <b>get</b> (int index)	Devuelve el elemento que hay en posición <i>index</i>
int <b>indexOf</b> (T elem)	Devuelve la posición de la primera vez que encuentra el objeto <i>elem</i>
int <b>lastIndexOf</b> (T elem)	Devuelve la posición de la última vez que aparece <i>elem</i>
ListIterator<T> <b>listIterator</b> ()	Devuelve un iterador para recorrer la lista de forma bi-direccional: hacia delante o atrás. Empezando en la posición 0
ListIterator<T> <b>listIterator</b> (int index)	Devuelve un iterador para recorrer la lista de forma bi-direccional: hacia delante o atrás. Empezando en la posición <i>index</i>

## ArrayList



### Métodos de inserción, eliminación, modif

Métodos para insertar, eliminar y modificar elementos	Significado
boolean <b>add</b> (T elem)	Añade el objeto <i>elem</i> al final de la lista
void <b>add</b> (int index, T elem)	Inserta el objeto <i>elem</i> en posición <i>index</i>
T <b>set</b> (int index, T elem)	Cambia el objeto en <i>index</i> por <i>elem</i> , y devuelve el objeto anterior
boolean <b>remove</b> (Object elem)	Elimina la primera ocurrencia del objeto <i>elem</i>
T <b>remove</b> (int index)	Elimina y devuelve el objeto en la posición <i>index</i>
void <b>clear</b> ()	Vacia la colección
boolean <b>addAll</b> (Collection<? extends T> col)	Añade todos los objetos de la colección <i>col</i> en la lista
boolean <b>addAll</b> (int index, Collection<? extends T> col)	Inserta todos los objetos de la colección <i>col</i> en la lista, empezando desde la posición <i>index</i>
boolean <b>removeAll</b> (Collection<? extends T> col)	Elimina las ocurrencias de todos los objetos de la colección <i>col</i>

## Ejemplos: ArrayList (1)



### Declaración, Creación, Inicialización, Modificación, Acceso

```
import java.util.List;
import java.util.ArrayList;
public class PruebaVectores2{
    public static void main(String[] args) {
        List<Integer> lista = new ArrayList<Integer>();
        lista.add(new Integer(1));
        lista.add(new Integer(2));
        lista.add(new Integer(3));
        lista.add(new Integer(3));
        lista.add(new Integer(3));
        lista.add(1, new Integer(4));
        lista.remove(1);
        lista.remove(new Integer(3));
        lista.set(2, new Integer(4));
        Integer i1 = lista.get(0);
        int entero1 = i1.intValue();
        System.out.println("el valor de la posición 0 es: "+entero1);
        int ind = lista.indexOf(new Integer(2));
        System.out.println("indice del numero 2: "+ind);
    }
}
```

Interfaz

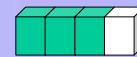
Clase concreta

1					
1	2				
1	2	3			
1	2	3	3	3	

// Declarar y crear

1	4	2	3	3	3
1	2	3	3	3	
1	2	3	3		
1	2	4	3		

## Ejemplos: ArrayList (2)



### Declaración, Creación, Inicialización, Recorrido

```
import java.util.List;
import java.util.ArrayList;
public class PruebaArrayList{
    public static void main(String args[]){
        List<Integer> lista = new ArrayList<Integer>();
        //crear objetos y añadir a la lista
        for (int i=1; i <= 5; i++)
            lista.add(i);

        //recorrido con un iterador
        Iterator<Integer> it = lista.iterator();
        while (it.hasNext() )
            System.out.println(it.next());

        //recorrido con "for-each"
        for (Integer ent: lista )
            System.out.println(ent);
    }
}
```

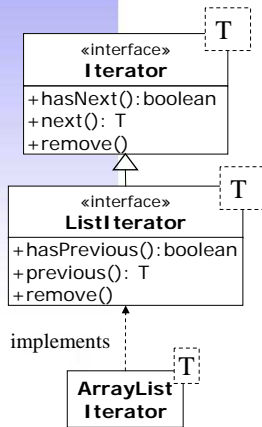
**Autoboxing:** el compilador sustituye el código con: `lista.add(new Integer(i));`

**El compilador sustituye el código "for-each" por el bucle anterior, que utiliza un iterador explícitamente**

**Unboxing:** el compilador sustituye el código con: `ent.intValue();`

## ArrayList (implementación)

### Iteradores



```

public interface Iterator<T>{
    /** Devuelve true si la iteración tiene mas elementos siguientes */
    public boolean hasNext();
    /** Devuelve el siguiente elemento de la iteración
     * @throws NoSuchElementException si se ha llegado al
     * final de la colección */
    public T next();
    /** Elimina el último elemento devuelto por la iteración
     * @throws IllegalStateException si se llama
     * a remove sin haber llamado primero a next */
    public void remove();
    /* si no lo implementa lanzamos UnsupportedOperationException
    */
}

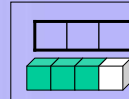
```

```

public interface ListIterator<T> extends Iterator<T> {
    /** Devuelve true si la iteración tiene mas elementos anteriores */
    public boolean hasPrevious();
    /** Devuelve el anterior elemento de la iteración
     * @throws NoSuchElementException si se ha llegado al
     * final de la colección */
    public T previous();
    ... }

```

## Array vs ArrayList



	Array	ArrayList
<b><i>Tipos de datos que almacena</i></b>	El que indiquemos al declararlo ( <code>int a[]</code> )	Elementos de tipo T ( <code>ArrayList&lt;T&gt; lista;</code> )
<b><i>Tamaño</i></b>	Mediante el <b>atributo</b> <code>length</code> ( <code>a.length</code> )	Mediante <b>métodos</b> <code>size()</code> y variable <code>capacity</code> ( <code>lista.size()</code> )
<b><i>Tamaño</i></b>	<b>fijo (inmutable)</b> indicado en el momento de su creación ( <code>a= new a[3]</code> )	<b>Puede crecer dinámicamente</b> si necesita más espacio ( <code>lista.add (new integer(4))</code> )
<b><i>Acceso, ins, elim, mod.</i></b>	Mediante el operador <b>indexación</b> <code>[]</code> ( <code>a[0]=2</code> )	Mediante métodos: <b>add</b> (T elem), <b>add</b> (int index, T elem), <b>remove</b> (Object elem)
<b><i>Está definido en</i></b>	El lenguaje Java	La biblioteca de clases del paquete <b>java.util</b>

## Listas Enlazadas

### LinkedList: ¿Qué son?



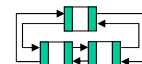
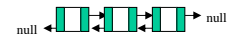
- Son estructuras de datos en las que los elementos **no se almacenan en posiciones contiguas** de memoria
- Se utilizan **para evitar movimientos** de grandes cantidades de datos pero tiene el coste añadido de una **referencia adicional** por cada elemento
- Cada elemento se almacena en un **nodo** que contiene:
  - una **referencia al objeto** que queremos almacenar y
  - una **referencia al siguiente nodo** en la lista
- También se mantiene una referencia al primer elemento de la lista (**top**)
- Se pueden añadir otros **métodos** que aporten funcionalidad adicional

31

## Listas Enlazadas

### clasificación

- Lista enlazada simple (**LinkedList**)
  - Acceso al **primer** nodo
  - Y de cada nodo enlace al **siguiente** nodo
- Lista con doble terminación (**DoubleEndedList**)
  - Igual que el LinkedList, pero con acceso también al **último** nodo
- Lista doblemente enlazada (**DoublyLinkedList**)
  - Acceso al **primer** y **último** nodo
  - De cada nodo enlace al **siguiente** y al **anterior**
- Lista circular (**CircularlyList**)
  - Acceso al **último** nodo
  - De cada nodo enlace al **siguiente** nodo
  - El **siguiente del último** nodo **es el primer** nodo

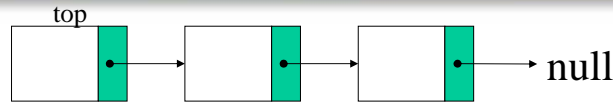


32



## Listas Enlazadas

### LinkedList: implementación

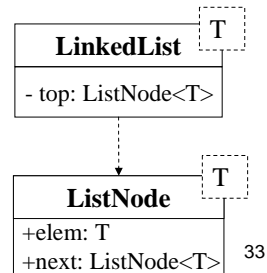


```

public class LinkedList<T> extends AbstractList<T>
    implements List<T>{
    private ListNode top;
}
  
```

```

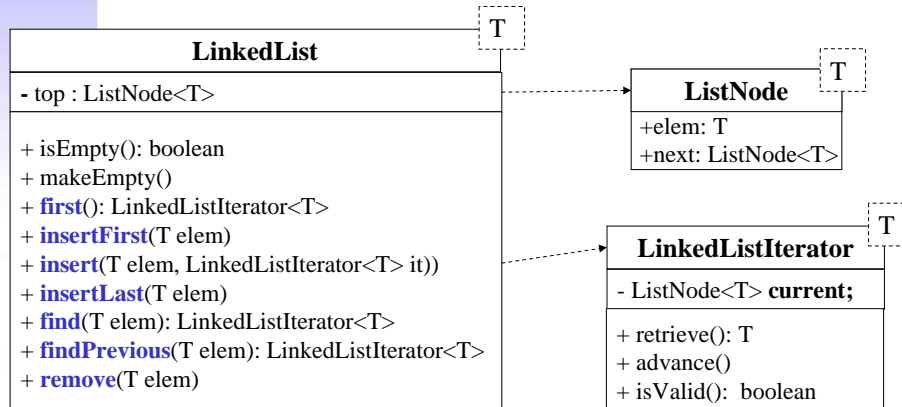
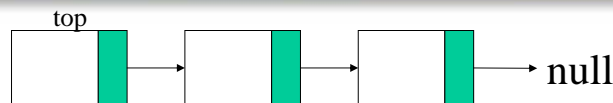
private class ListNode<T>{
    public T elem;
    public Node<T> next;
    public Node(T pElem){
        this.elem = pElem;
    }
}
  
```



33

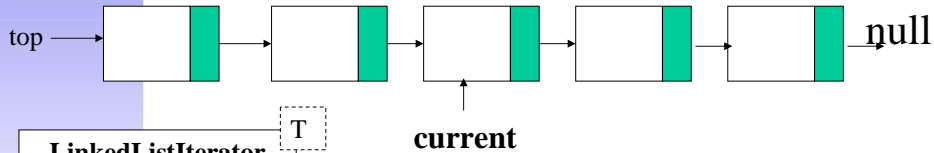
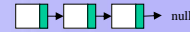
## Listas Enlazadas

### LinkedList: implementación



## Listas Enlazadas

### LinkedListIterator: La clase iteradora



LinkedListIterator
- <b>current</b> : ListNode<T>
+ isValid(): boolean
+ retrieve(): T
+ advance()

```

public class LinkedListIterator<T> implements ...{
    private ListNode current;
    /** Devuelve true si la iteración tiene mas elementos
     * siguientes */
    public boolean isValid(){...}
    /** Devuelve el elemento actual
     * @throws NoSuchElementException si se ha llegado al
     * final de la colección */
    public T retrieve(){...}
    /** Avanza el cursor 'current' al siguiente */
    public void advance(){...}
}
    
```

## Listas Enlazadas

### LinkedListIterator: La clase iteradora

```

import EstructurasDeDatos.LinkedList;
import EstructurasDeDatos.LinkedListIterator;
public class PruebaLinkedList{
    public static void main(String args[]){
        LinkedList<Integer> lista = new LinkedList<Integer>();

        //crear objetos y añadir a la lista
        lista.insertFirst( new Integer(1) );
        LinkedListIterator<Integer> it = lista.first();
        for(int i=2; i<=5; i++){
            lista.insert( new Integer(i), it); // el nuevo se inserta después del cursor current y
                                                // posterior el cursor se mueve al último insertado

            //buscar el número 3
            LinkedListIterator<Integer> it2 = lista.find( new Integer(3));

            //muestra en pantalla el número 3 y los siguientes a él
            while (it2.isValid()){
                System.out.println(it2.retrieve());
                list.advance();
            }
        }
    }
}
    
```

36

```

graph LR
    Node1[ ] --> Node2[ ]
    Node2 --> Node3[ ]
    Node3 --> null[null]
    style Node1 fill:#fff,stroke:#333,stroke-width:1px
    style Node2 fill:#fff,stroke:#333,stroke-width:1px
    style Node3 fill:#fff,stroke:#333,stroke-width:1px
    style null fill:none,stroke:none
  
```



## 38

## Listas (List)

### ArrayList vs. LinkedList (comparing costs)

- **Test1:** Crear una lista (`List`) de números enteros añadiendo elementos al **final**.

```
public static void makeList1(List<Integer> list,int N)
{
    list.clear();
    for(int i=0; i < N; i++)
        list.add(i);
}
```

39

## Listas (List)

### ArrayList vs. LinkedList (comparing costs)

- **Test2:** Crear una lista (`List`) de números enteros añadiendo elementos al **inicio**.

```
public static void makeList2(List<Integer> list,int N)
{
    list.clear();
    for(int i=0; i < N; i++)
        list.add(0,i);
}
```

40

## Listas (List)

### ArrayList vs. LinkedList (comparing costs)

- **Test3:** Sumar los números de una lista (`List`).

```
public static int sum(List<Integer> list)
{
    int total = 0;
    for(int i=0; i < list.size(); i++)
        total += list.get(i);
}
```

41

## Listas Enlazadas

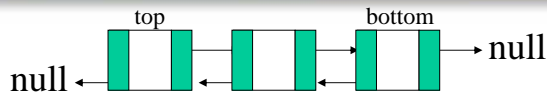
### ArrayList vs. LinkedList

- |  |  |
|--|--|
| <ul style="list-style-type: none"><li>• <b><u>Desventajas:</u> <math>O(n)</math></b><ul style="list-style-type: none"><li>– Insertar un elemento al inicio o en el medio</li><li>– Longitud fija, coste al crecer la lista</li></ul></li></ul> | <ul style="list-style-type: none"><li>• <b><u>Ventajas:</u></b><ul style="list-style-type: none"><li>– Insertar elemento <math>O(1)</math></li><li>– Crece hasta terminar la memoria</li></ul></li></ul>                                       |
| <ul style="list-style-type: none"><li>• <b><u>Ventajas:</u> <math>O(1)</math></b><ul style="list-style-type: none"><li>– Buscar elemento en la posición <math>i</math></li><li>– Obtener el elemento anterior al actual</li></ul></li></ul>    | <ul style="list-style-type: none"><li>• <b><u>Desventajas:</u> <math>O(n)</math></b><ul style="list-style-type: none"><li>– Buscar elemento en la posición <math>i</math></li><li>– Obtener el elemento anterior al actual</li></ul></li></ul> |

42

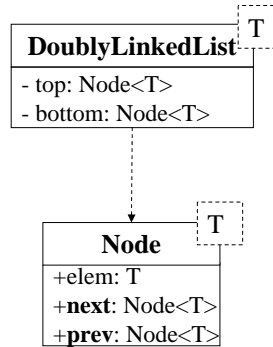
## Listas Doblemente Enlazadas

### DoublyLinkedList: ¿Qué son?



```
public class DoublyLinkedList<T>{
    private Node<T> top;
    private Node<T> bottom;
}
```

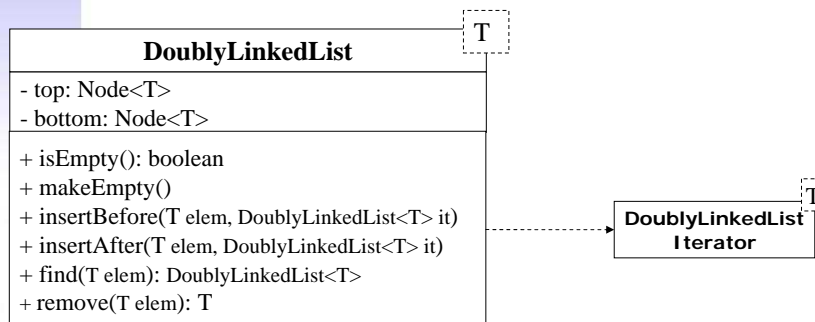
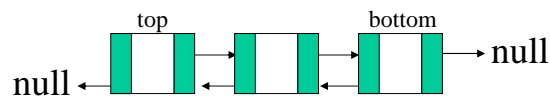
```
private class Node{
    public T elem;
    public Node<T> next;
    public Node<T> prev;
    public Node(T pElem){
        this.elem = pElem;
    }
}
```



43

## Listas Doblemente Enlazadas

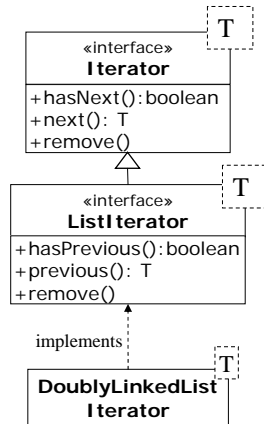
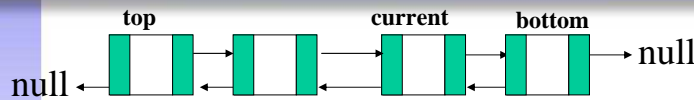
### DoublyLinkedList: ¿Qué son?



44

## Listas doblemente enlazadas

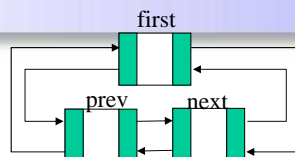
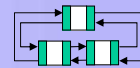
### DoubleLinkedListIter: la clase iteradora



45

## Listas Circulares

### ¿Qué son?



```

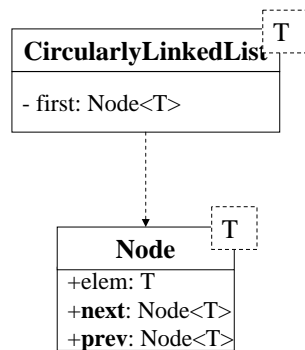
public class CircularlyLinkedList<T>{
    private Node<T> first;
}

```

```

public class Node<T>{
    public T elem;
    public Node<T> next;
    public Node<T> prev;
    public Node(T pElem){
        this.elem = pElem;
    }
}

```



47

## Listas Enlazadas

### LinkedList vs Vectores

	ArrayList	LinkedList
<i>Tipos de datos que almacena</i>	Almacena <b>elementos del tipo que declaremos</b> en la clase node	<ul style="list-style-type: none"> <li>Almacena <b>elementos del tipo que declaremos</b> en la clase node</li> <li>Almacena también <b>referencias</b> a los elementos consecutivos</li> </ul>
<i>Tamaño</i>	<ul style="list-style-type: none"> <li>Mediante <b>métodos</b></li> <li>Puede crecer dinámicamente</li> </ul>	<ul style="list-style-type: none"> <li>Mediante métodos,</li> <li>Puede crecer dinámicamente</li> <li><b>Consume espacio adicional</b> porque también almacena <b>referencias</b> a otros elementos</li> </ul>
<i>Acceso, inserción, eliminación, modificación</i>	<ul style="list-style-type: none"> <li>Mediante métodos: add (T obj), remove (T obj)</li> </ul>	<ul style="list-style-type: none"> <li>Utiliza clase auxiliar ListNode&lt;T&gt;</li> <li>Mediante métodos: add (T obj), remove (T obj)</li> </ul>

50

## Conclusiones

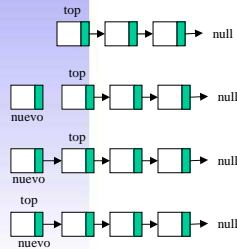
- Los **arrays** y los **ArrayList** son más eficientes para **acceder** a elementos **de forma aleatoria**
  - Utilizamos **arrays** cuando
    - Tenemos un *nº conocido de elementos*
    - Todos los elementos son del mismo tipo: *primitivo o objeto*
  - Utilizamos **ArrayList** cuando
    - Los elementos pueden ser del tipo o subtipo definido (objeto)
    - El nº de elementos varia (*tamaño dinámico*)
- Las **listas enlazadas** (simples o doblemente) son más eficientes cuando se producen **muchas inserciones y eliminaciones de elementos** en posiciones arbitrarias
- Los **ArrayList** y las **listas doblemente enlazadas** son mejores en el recorrido bidireccional que las **listas enlazadas simples**

51

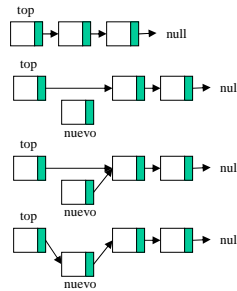


## Apéndice I:

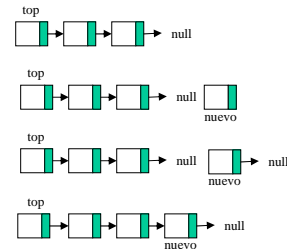
### Insertar un nodo en LinkedList



insertFirst()  
prepend()



insertAt (int index)

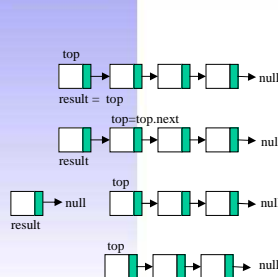


insertLast()  
append()

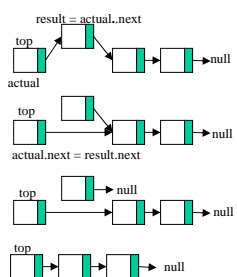
52

## Apéndice II:

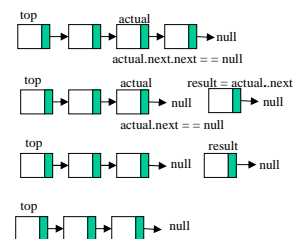
### Eliminar un nodo en LinkedList



removeFirst()



removeAt(int index)



removeLast()

resultado

53

## Comparativas

- **Vector vs ArrayList vs HashSet**

- [http://www.theserverside.com/news/thread.tss?track=NL-461&ad=782580HOUSE&thread\\_id=60728&asrc=EM\\_NLN\\_12321103&uid=2726991](http://www.theserverside.com/news/thread.tss?track=NL-461&ad=782580HOUSE&thread_id=60728&asrc=EM_NLN_12321103&uid=2726991)
- <http://www.javacodegeeks.com/2010/08/java-best-practices-vector-arraylist.html>

- **Java Collections**

- <http://sctchpad.wordpress.com/2008/10/21/java-collections-performance-benchmark/>
- La clase `java.util.LinkedList` implementa una lista **doblemente enlazada**

54

## Otras implementaciones

- **JDK 1.4.2**

- <http://download.oracle.com/javase/1.4.2/docs/api/java/util/List.html>
- <http://download.oracle.com/javase/1.4.2/docs/api/java/util/ArrayList.html>
- <http://download.oracle.com/javase/1.4.2/docs/api/java/util/LinkedList.html>

- <http://www.java-tips.org/java-se-tips/java.lang/linked-list-implementation-in-java.html>

55

## Ejemplos de utilización

- <http://www.java-tips.org/java-se-tips/java.util/how-to-use-list-interface.html>
- <http://www.java-tips.org/java-se-tips/java.lang/use-of-arraylist-class.html>
- <http://www.java-tips.org/java-se-tips/java.util/how-to-create-a-linked-list.html>

56