

Java

Estructuras de datos

Facundo Galán

A decorative blue wavy line that starts from the left edge of the slide and curves upwards and to the right, ending near the bottom right corner.

ArrayList

- Nos permite tener los elementos indexados, y a su vez insertar elementos al final de la estructura en $O(1)$.

```
ArrayList<Integer> arr = new ArrayList<Integer> ();
```

```
arr.size() // Devuelve 0
```

```
arr.add(5); // Agrega 5 al final
```

```
arr.size() // Devuelve 5
```

```
arr.clear() // Vacía el arreglo
```

<http://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html>

ArrayList (Cont.)

- Para ordenarlo y hacer búsquedas binarias, no es necesario implementarlas nosotros (en los casos más comunes al menos).

```
ArrayList<Integer> arr = new ArrayList<Integer> ();
```

```
int [] arr2 = new int[105];
```

```
Array.sort(arr);
```

```
Array.sort(arr2);
```

```
binary_search(arr2, 2);
```

<http://docs.oracle.com/javase/7/docs/api/java/util/Arrays.html>

Queue

- Una cola dinámica ya implementada.

```
Queue<Integer> q = new Queue<Integer>();
```

```
q.add(4);
```

```
q.peek();
```

```
q.poll();
```

```
q.size();
```

```
q.isEmpty();
```

<http://docs.oracle.com/javase/7/docs/api/java/util/Queue.html>

Stack

- Una pila dinámica ya implementada.

```
Stack<Integer> stk = new Stack<Integer>();
```

```
stk.push(4);
```

```
stk.pop();
```

```
stk.poll();
```

```
stk.size();
```

```
stk.isEmpty();
```

<http://docs.oracle.com/javase/7/docs/api/java/util/Stack.html>

Deque

- Una deque es una estructura que funciona como una cola y como una pila (permite inserciones y extracciones tanto de un lado como del otro en $O(1)$).

```
Deque<Integer> dq = new Deque<Integer>();  
dq.addFirst(4);  
dq.addLast(5);  
dq.pollFirst();  
dq.pollLast();  
dq.size();
```

<http://docs.oracle.com/javase/7/docs/api/java/util/Deque.html>

HashSet

- Estructura para manejar conjuntos (no permite repetidos). Las operaciones son de $O(1)$, y los elementos no están ordenados.

```
HashSet<Integer> hs = new HashSet<Integer>();
```

```
hs.add(5);
```

```
hs.contains(10);
```

```
hs.remove(4);
```

<http://docs.oracle.com/javase/7/docs/api/java/util/HashSet.html>

TreeSet

- Estructura para manejar conjuntos (no permite repetidos). Las operaciones son de $O(\log(N))$, y los elementos están ordenados.

```
TreeSet <Integer> ts = new TreeSet <Integer>();
```

```
ts.add(5);
```

```
ts.floor(7);
```

```
ts.contains(10);
```

```
ts.remove(4);
```

<http://docs.oracle.com/javase/7/docs/api/java/util/TreeSet.html>

LinkedHashSet

- Estructura con el mismo comportamiento de un HashSet, con la cualidad de que sus elementos pueden ser recorridos en el orden ingresado.

```
LinkedHashSet <Integer> lhs;
```

```
lhs = new LinkedHashSet <Integer>();
```

```
lhs.add(5);
```

```
lhs.contains(10);
```

```
lhs.remove(4);
```

<http://docs.oracle.com/javase/7/docs/api/java/util/LinkedHashSet.html>

HashMap

- Estructura del tipo diccionario (clave, valor). Permite almacenar elementos, y accederlos por su clave. Las operaciones son en $O(1)$, y no se mantiene ordenado.

```
HashMap <Integer, Integer> hm;
```

```
hm = new HashMap <Integer, Integer>();
```

```
hm.put(5, 6);
```

```
hm.containsKey(10); //  $O(1)$ ;
```

```
hm.containsValue(10); //  $O(N)$ 
```

<http://docs.oracle.com/javase/7/docs/api/java/util/HashMap.html>

TreeMap

- Estructura del tipo diccionario (clave, valor). Permite almacenar elementos, y accederlos por su clave. Las operaciones son $O(\log(N))$, y se mantiene ordenado.

```
TreeMap <Integer, Integer> tm;
```

```
tm = new TreeMap <Integer, Integer>();
```

```
tm.ceilingKey (10); //  $O(\log(N))$ ;
```

```
tm.containsKey(10); //  $O(\log(N))$ ;
```

```
tm.containsValue(10); //  $O(N)$ 
```

<http://docs.oracle.com/javase/7/docs/api/java/util/TreeMap.html>

LinkedHashMap

- Estructura con el mismo comportamiento de un HashMap, con la cualidad de que sus elementos pueden ser recorridos en el orden ingresado.

```
LinkedHashSet <Integer> lhs;
```

```
lhs = new LinkedHashSet <Integer>();
```

```
hm.put(5, 6);
```

```
hm.containsKey(10); // O(1);
```

```
hm.containsValue(10); // O(N)
```

<http://docs.oracle.com/javase/7/docs/api/java/util/LinkedHashMap.html>

PriorityQueue

- Estructura del tipo heap. Por defecto se comporta como una minheap, pero se le puede redefinir el comparador.

```
PriorityQueue<Integer> pq;  
pq = new PriorityQueue<>();  
pq = new PriorityQueue<>(9, Collections.reverseOrder());  
pq.add(9);  
pq.poll();
```

<http://docs.oracle.com/javase/7/docs/api/java/util/PriorityQueue.html>

Recorrido de las estructuras

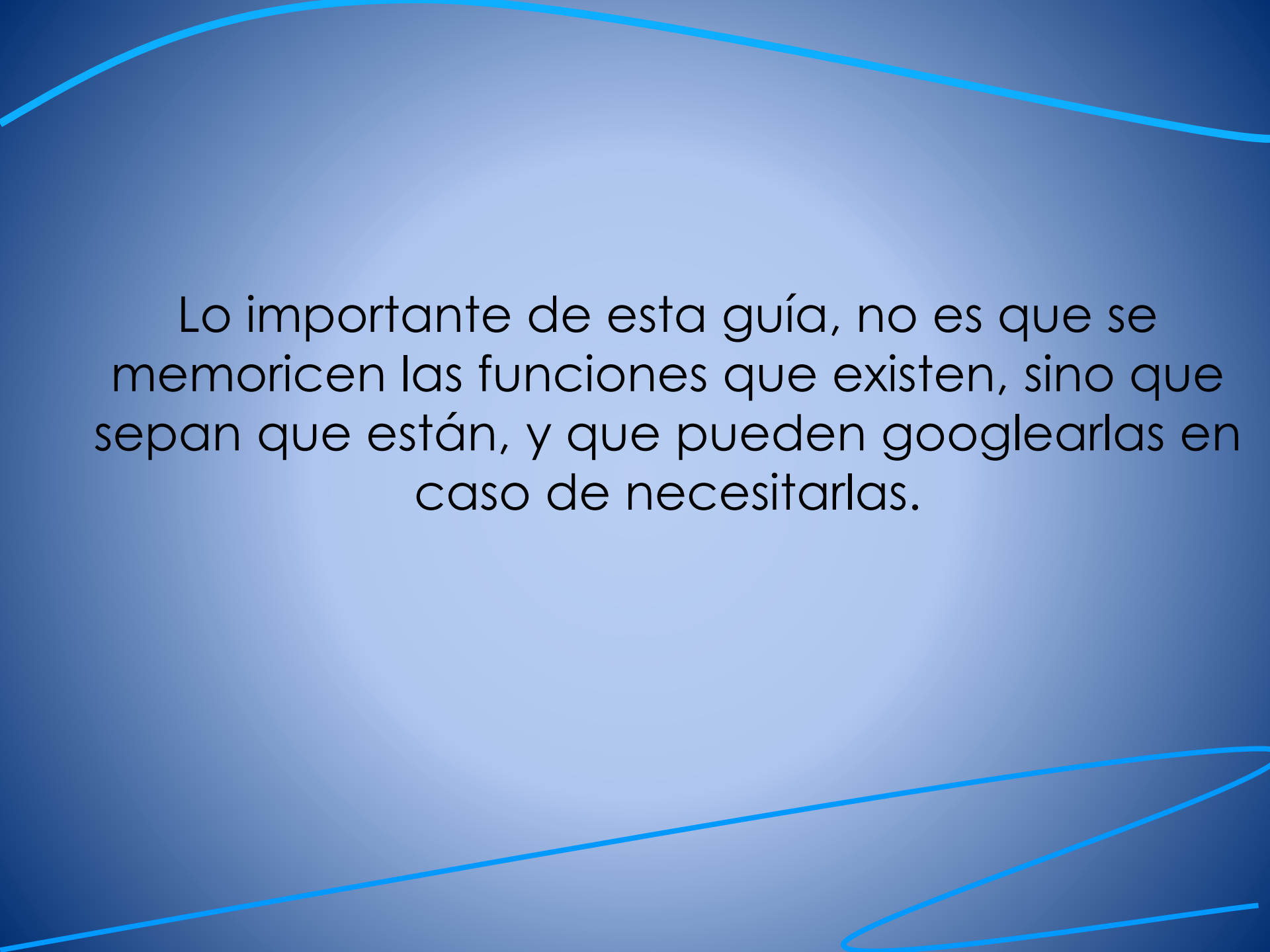
- Se pueden recorrer de dos formas (siendo str cualquier estructura):

(1)

```
Iterator<Integer> itr = str.iterator();  
while (itr.hasNext()){  
    System.out.println(itr.next());  
}
```

(2)

```
for (Integer x: str) {  
    System.out.println(x);  
}
```

Lo importante de esta guía, no es que se memoricen las funciones que existen, sino que sepan que están, y que pueden googlearlas en caso de necesitarlas.