CLASE ESPEJO

Estructura de datos.

Mtro. Jesús Abraham Zazueta Castillo.

LinkedIn
Jesús Abraham Zazueta Castillo.

GitHub
Jesús Abraham Zazueta Castillo

CORHUILA
CORPORACIÓN UNIVERSITARIA DEL HUILA
Vigilada Mineducación

CORPORACIÓN UNIVERSITARIA DEL HUILA - CORHUILA
"Diseño y prestación de servicios de docencia, investigación y extensión de programas de pregrado, aplicando todos los requisitos de las normas ISO implementadas en sus sedes Neiva y Pitalito"

ITSON
UNIVERSIDAD

Algoritmos y Programación

LinkedIn
Jesús Abraham Zazueta Castillo

GitHub
Jesús Abraham Zazueta Castillo

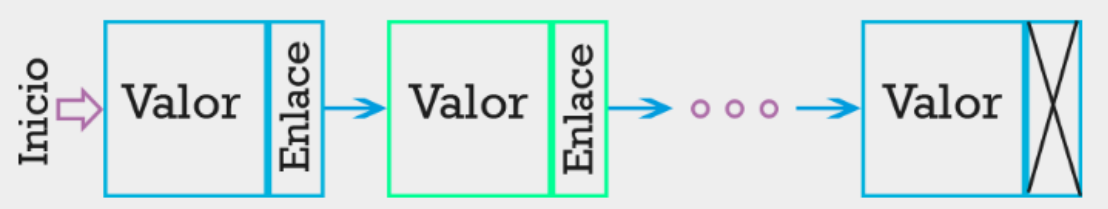
Tipos de Datos Abstractos

- Es un modelo que define **valores** y las **operaciones** que se pueden realizan sobre ellos. Y se denomina abstracto ya que la intención es que quien lo utiliza, no necesita conocer los detalles de la representación interna o bien el cómo están implementadas las operaciones.

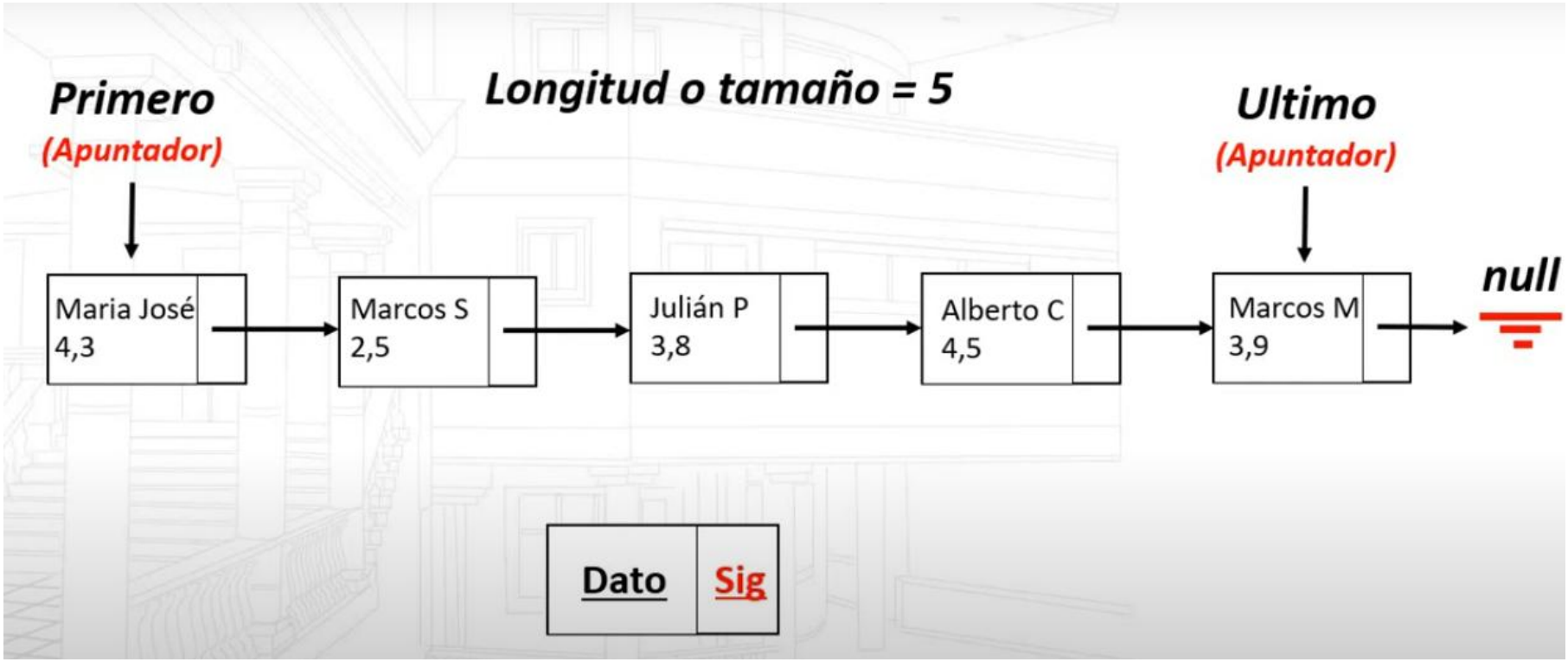
TDAs Clasificaciones

Lista enlazada

- Una lista es una estructura de datos lineal que se puede representar simbólicamente como un conjunto de nodos enlazados entre sí.
- Una lista es una secuencia de elementos del mismo tipo de cada uno de los cuales se puede decir cual es su siguiente (en caso de existir).



Lista enlazada



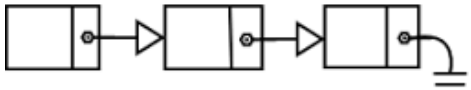
Lista enlazada

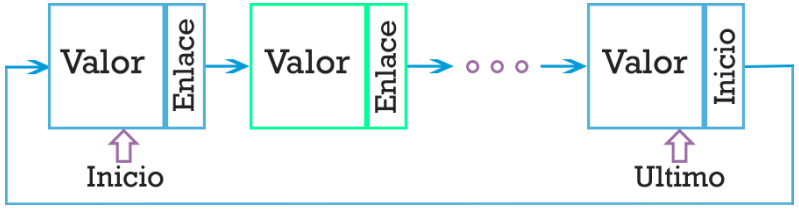
- Operaciones:
  - Alta – agregar un nodo
  - Baja – desvincular un nodo
  - Modificar – Cambiar un nodo de la lista
  - Búsqueda – Buscar un valor en la lista
  - Recorrer (Listar) – Recorrer e imprimir los valores
  - Indicar el tamaño de la lista

Lista enlazada

- Lista enlazada simple
- Lista circulares

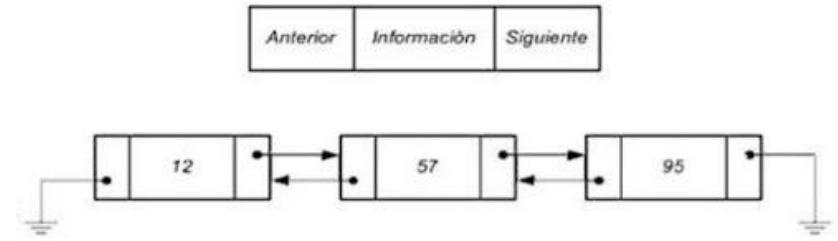
Estructura de un nodo

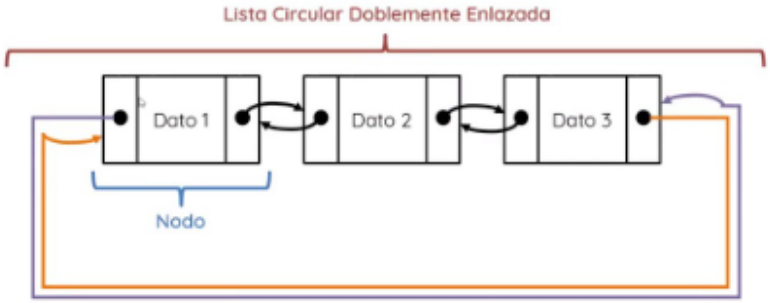

Lista enlazada



Lista enlazada

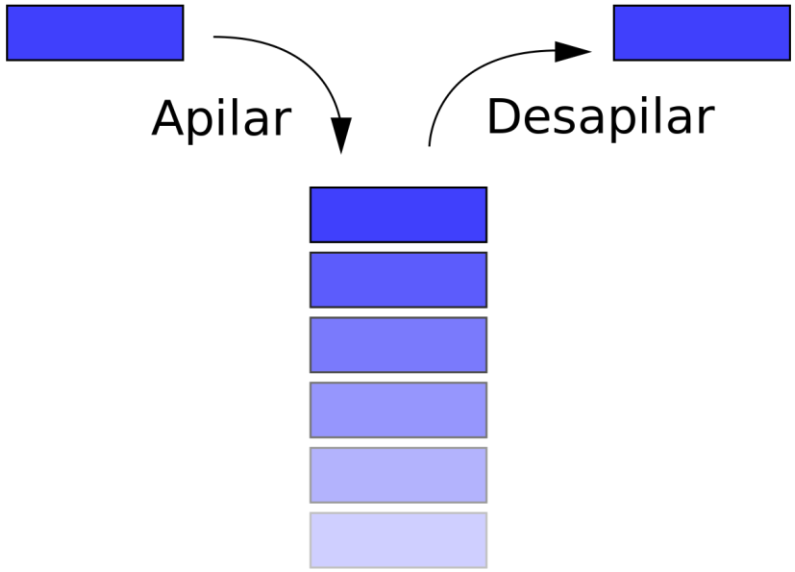
- Lista doblemente enlazada
- Lista circulares doblemente enlazada





Pila

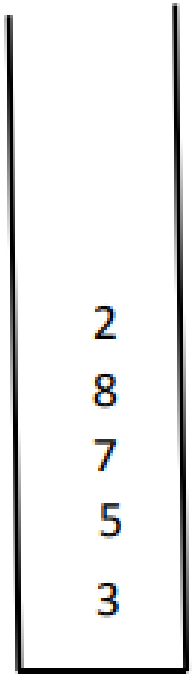
- Colección lineal de objetos actualizada en un extremo llamado tope usando una política LIFO (last-in first-out, el primero en entrar es el último en salir).

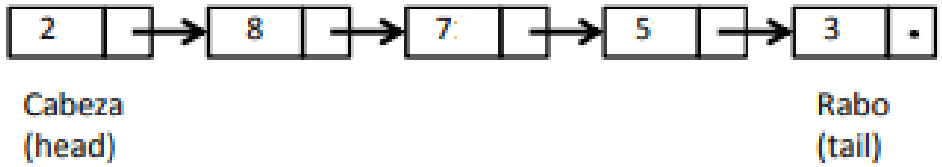


Pila - Operaciones

- **push(e) Apilar**: Inserta el elemento e en el tope de la pila
- **pop() Desapilar:** Elimina el elemento del tope de la pila y lo entrega como resultado. Si se aplica a una pila vacía, produce una situación de error.
- **isEmpty():** Retorna verdadero si la pila no contiene elementos y falso en caso contrario.
- **top():** Retorna el elemento del tope de la pila. Si se aplica a una pila vacía, produce una situación de error.
- **size():** Retorna un entero natural que indica cuántos elementos hay en la pila.

Pila



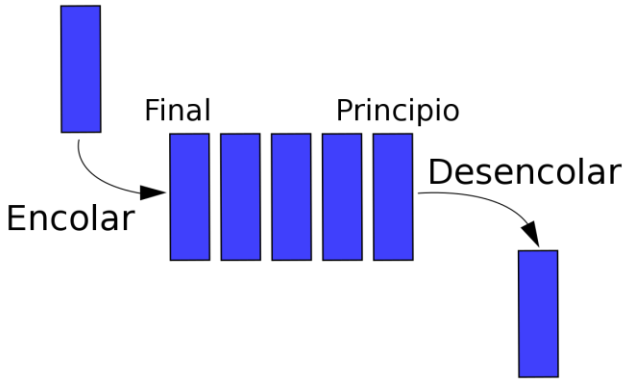


Ejemplo:

- Inventario por lotes de materiales de construcción.

Cola

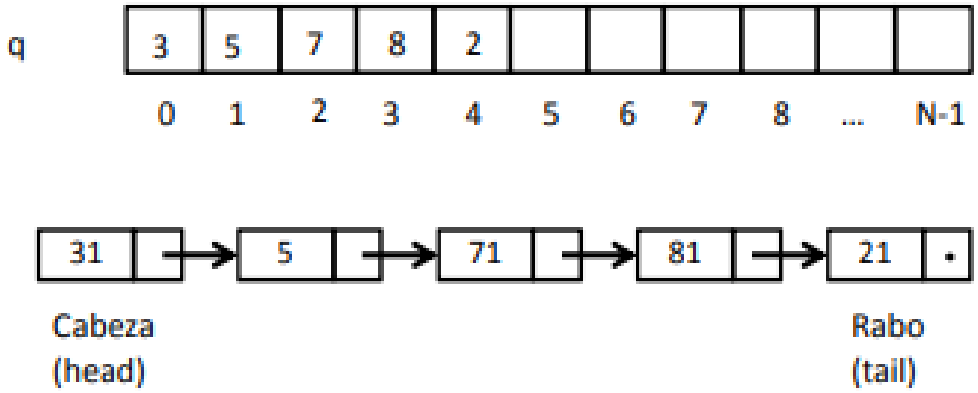
- Colección lineal de objetos actualizada en sus extremos llamados frente y rabo siguiendo una política FIFO (first-in first-out, el primero en entrar es el primero en salir) (También se llama FCFS = First-Come First-Served).



Cola - Operaciones

- **enqueue(e) Encolar:** Inserta el elemento en el rabo/final de la cola
- **dequeue() Desencolar:** Elimina el elemento del frente de la cola y lo retorna. Si la cola está vacía se produce un error.
- **front():** Retorna el elemento del frente de la cola. Si la cola está vacía se produce un error.
- **isEmpty():** Retorna verdadero si la cola no tiene elementos y falso en caso contrario.
- **size():** Retorna la cantidad de elementos de la cola.

Cola

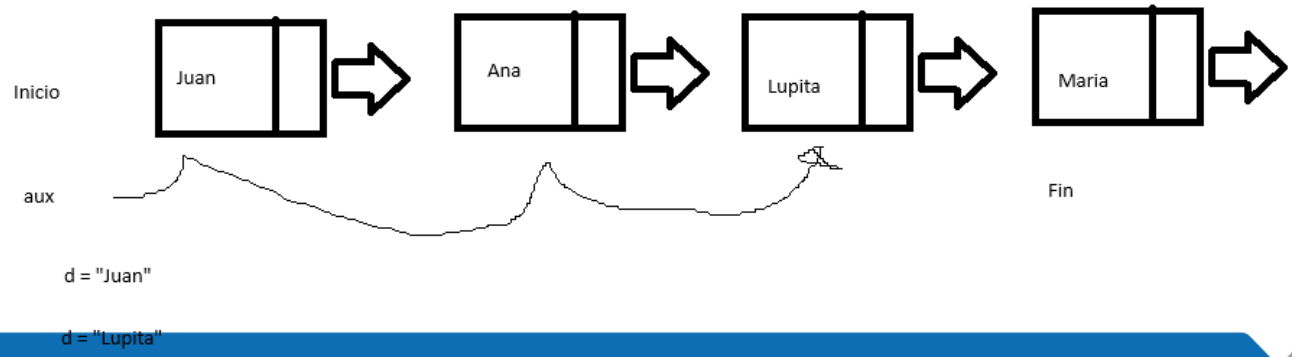


Ejemplo:

- Sistema de turnos en el banco

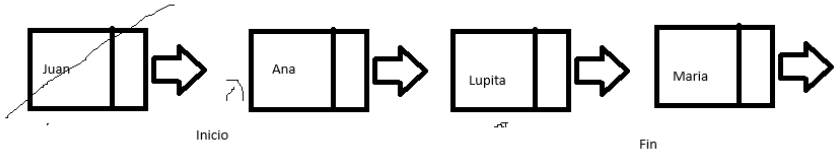
Buscar un elemento en la lista

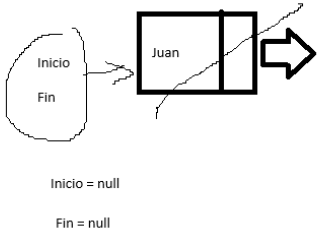
```
public boolean buscarDato(String d) {
    boolean encontrado = false;
    Nodo aux = inicio;
    while(encontrado != true && aux != null) {
        if(d.equals(aux.valor)) {
            encontrado = true;
            break;
        }else{
            aux = aux.siguiente;
        }
    }

    return encontrado;
}
```



Eliminar un elemento al inicio

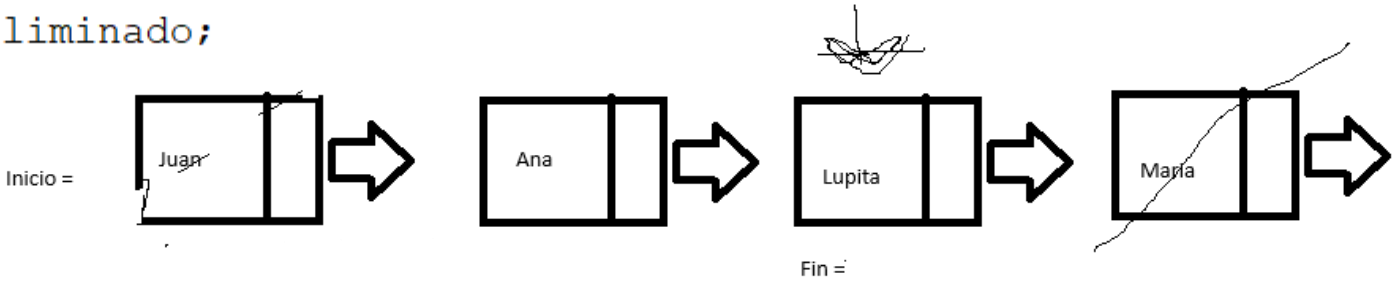
```
public String eliminarDatoInicio() {
    String eliminado = inicio.valor;
    int contador = cantidadNodos();
    if(contador == 1){
        inicio = null;
        fin = inicio;
    }else{
        inicio = inicio.siguiente; //inicio.s
    }

    return eliminado;
}
```



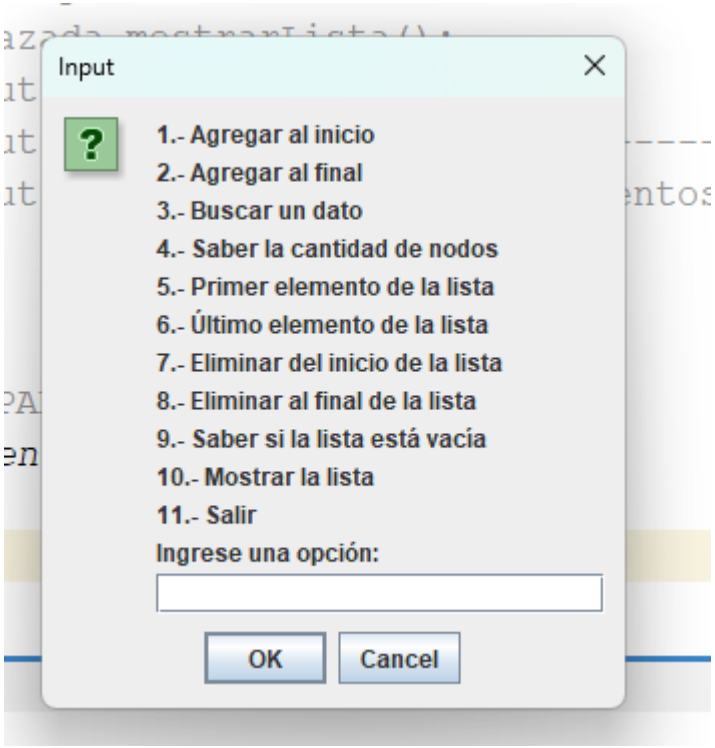


Eliminar un elemento al final

```
public String eliminarDatoFin() {
    String eliminado = fin.valor;
    Nodo aux = inicio;
    while(aux.siguiente != fin) {
        aux = aux.siguiente;
    }

    fin = aux;
    fin.siguiente = null;

    return eliminado;
}
```



Menú de opciones



Ordenamiento y búsqueda

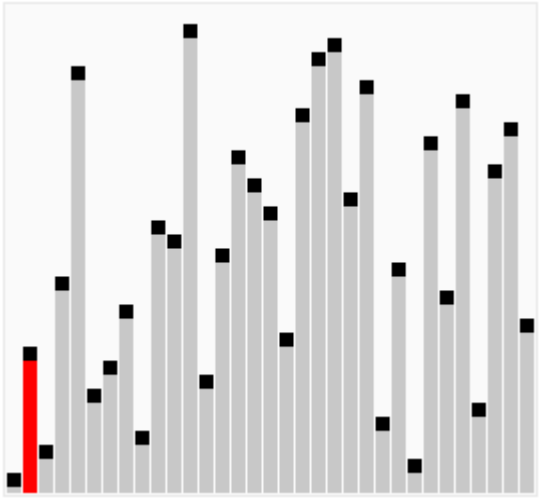
- [1, 5, 6, 9, 10, 21, 3, 100]
- [1, 5, 21, 9, 10, 21, 3, 10]

Condiciones búsqueda*

No se repitan datos

El arreglo este ordenado previamente

Algoritmos de ordenamiento



- Ordenamiento burbuja
- Ordenamiento por selección
- Ordenamiento por inserción

10	4	8	5	12	2	6	11	3	9	7	1

Bubble Sort:

3	6	2	7	1	8

Algoritmos de búsqueda

- Búsqueda secuencial
- Búsqueda binaria

Asignación - Investigar cada algoritmo

- Condiciones para el algoritmo
- Complejidad
- Pseudocódigo
- Ejemplos de programación e implementación