Universidad Nacional Mayor de San Marcos
Universidad del Perú. Decana de América

Estructura de Datos

Semana 5

Universidad Nacional Mayor de San Marcos
Universidad del Perú. Decana de América

Logro de la sesión

Al finalizar la sesión, el estudiante:

- **Identifica, analiza y resuelve problemas algorítmicos que usan el tipo abstracto de datos: pilas y colas, desarrollando métodos que usen esas estructuras.**

Estructuras de datos lineales

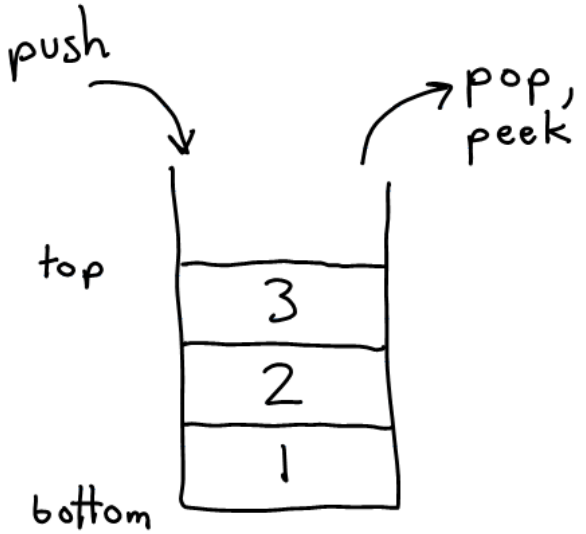
TAD Pila y TAD Cola

01

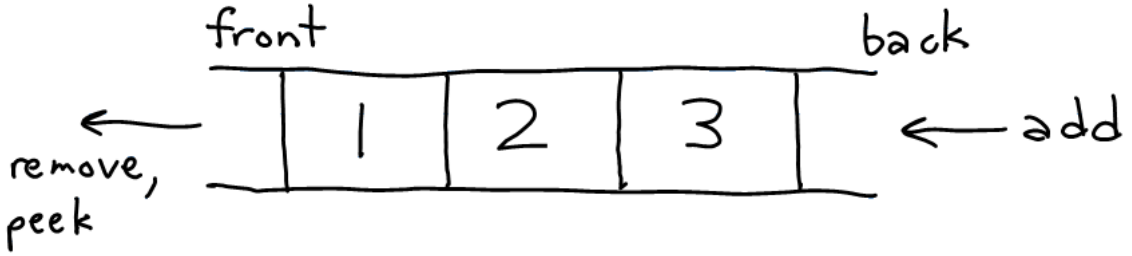
Pilas y colas

- A veces es bueno tener una colección que sea menos potente, pero que esté optimizada para realizar ciertas operaciones muy rápidamente.

- Hoy examinaremos dos colecciones especiales:
  - **pila** : Recupera elementos en el orden inverso al que fueron agregados.
  - **cola** : recupera los elementos en el mismo orden en que se agregaron.



pila



cola

Tipos abstractos de datos (TAD, ADT)

- **tipos abstractos de datos (TAD, ADT)** : una especificación de una colección de datos y las operaciones que se pueden realizar en ellos.
  - Describe *lo que* hace una colección, no *cómo* lo hace.

- No sabemos exactamente cómo se implementa una pila o cola, y no es necesario.
  - Solo necesitamos entender la idea de la colección y qué operaciones puede realizar.

(Las pilas generalmente se implementan con arreglos, pero también con listas enlazadas; las colas a menudo se implementan usando una lista enlazada).

Estructuras de datos lineales

- **Especificación formal:**

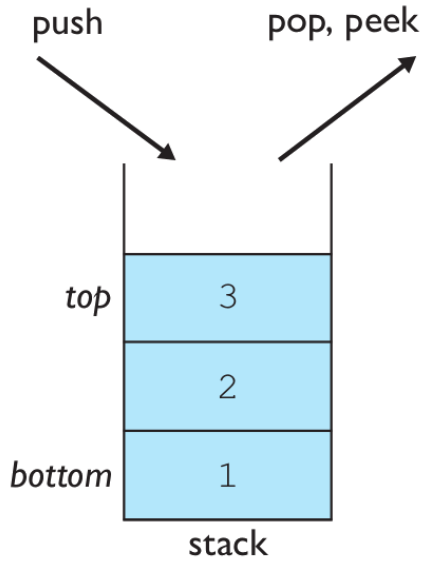
```
public interface IStack {

    public boolean isEmpty();
    public void push(Integer elem);
    public Integer pop();
    public Integer top();
    public int getSize();

}
```

Pilas

- **pila** : Una colección basada en el principio de agregar elementos y recuperarlos en el orden inverso.
  - Último en entrar, primero en salir ("LIFO")
  - Los elementos se almacenan en orden de inserción, pero no pensamos que tengan índices.
  - El cliente solo puede agregar/quitar/examinar el último elemento agregado (el de "arriba").



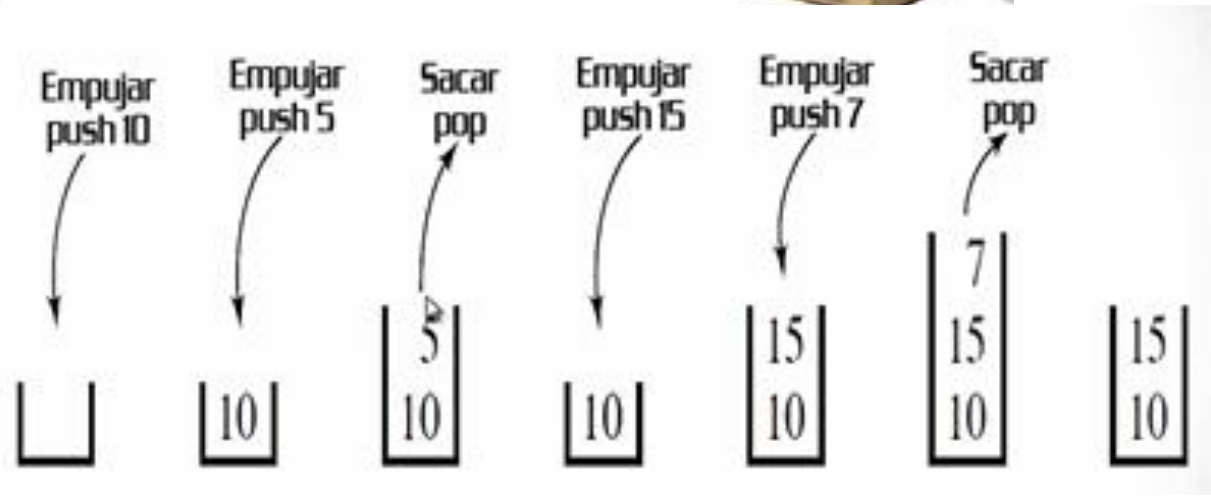
- operaciones básicas de una pila:
  - **push** : Añade un elemento a la parte superior.
  - **pop** : elimina el elemento superior.
  - **peek** : examina el elemento superior.

Pilas - Características

Una Pila (stack) es una colección ordenada de elementos a los cuales sólo se puede acceder por un único lugar o extremo de la pila. Los elementos se añaden o se quitan (borran) de la pila sólo por su parte superior (cima). Este es el caso de una pila de platos, una pila de libros, etc.

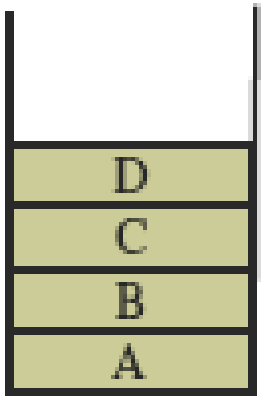
Operaciones Básicas de una pila

1. Crear Pila
2. Insertar Dato
3. Quitar Dato
4. Pila Vacía
5. Pila Llena
6. Limpiar Pila
7. Cima Pila
8. Tamaño de la Pila



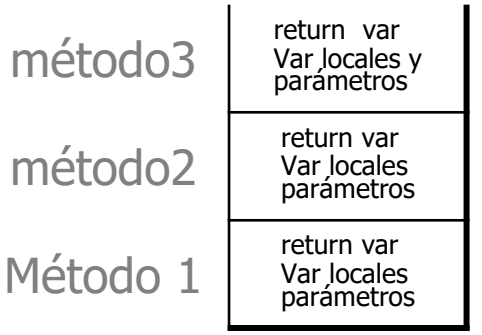
Estructura de datos LIFO: Last In First Out
(último en entrar, primero en salir)

No se pueden extraer los elementos C, B y A sin antes extraer D.

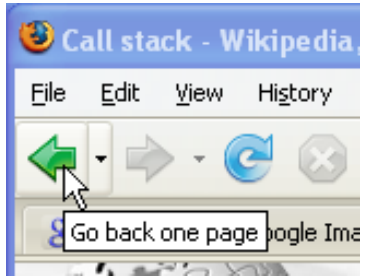


Pilas en informática

- Lenguajes de programación y compiladores:
  - Las llamadas a métodos/procedimientos se colocan en una pila (call=push, return=pop)
  - Los compiladores usan pilas para evaluar expresiones (sintaxis, anidamiento)



- Emparejar pares de cosas relacionadas:
  - averiguar si un string es un palíndromo
  - examinar un archivo para ver si sus llaves { } y otros operadores se corresponden
  - convertir expresiones "infija" a "postfija" o "prefija"



- Algoritmos sofisticados:
  - buscando a través de un laberinto con “backtracking"
  - muchos programas utilizan una "pila deshacer" de operaciones anteriores

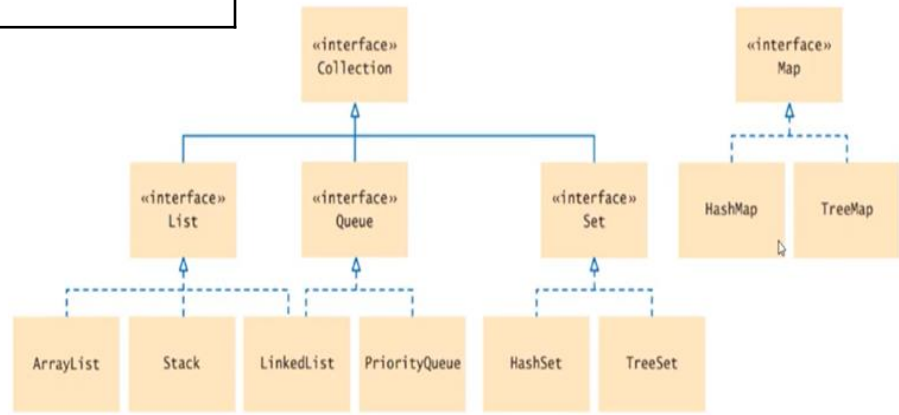
Clase Stack

`Stack<`**E**`>()`	construye una nueva pila con elementos de tipo **E**
`push(`**valor**`)`	coloca el valor dado en la parte superior de la pila
`pop()`	elimina el valor superior de la pila y lo devuelve; lanza `la excepción EmptyStackException` si la pila está vacía
`peek()`	devuelve el valor superior de la pila sin eliminarlo; lanza `la excepción EmptyStackException` si la pila está vacía
`size()`	devuelve el número de elementos en la pila
`isEmpty()`	devuelve `true` si la pila no tiene elementos

```
Stack<Integer> s = new Stack<Integer>();
s.push(42);
s.push(-3);
s.push(17); // abajo [42, -3, 17] arriba

System.out.println(s.pop()); // 17
```

- `Stack` tiene otros métodos, pero no recomendamos usarlos.



Interfaces y clases en Java Collection

Pilas mediante la clase Stack (java.util.Stack)

```
package tipoPila;
import java.util.Stack;
public class PilaStack {
    public static void main(String[] args) {
        Stack pilon=new Stack();
        pilon.push(10);
        pilon.push(20);
        pilon.push(30);
        pilon.push(40);
        pilon.push(50);
        pilon.push(333);
        System.out.println("El tamaño de la pila es "+pilon.size());
        System.out.println("La cima es "+pilon.peek());
        System.out.println("Sacando un elemento de la pila "+ pilon.pop());
        System.out.println("Sacando un elemento de la pila "+ pilon.pop());
        System.out.println("El tamaño de la pila es "+pilon.size());
        System.out.println("La pila está vacía?"+pilon.isEmpty());
    }

}
```

```
run:
El tamaño de la pila es 6
La cima es 333
Sacando un elemento de la pila 333
Sacando un elemento de la pila 50
El tamaño de la pila es 4
La pila está vacía?false
BUILD SUCCESSFUL (total time: 0 seconds)
```

Pilas mediante la clase Stack - Ejercicio

Asuma una pila simple para enteros.

```
Stack s = new Stack();
s.push(12);
s.push(4);
s.push( s.peek() + 2 );
s.pop()
s.push( s.peek() );
//Cuál es el contenido de la pila?
```

Pilas mediante la clase Stack – Error común

Escriba un método para imprimir el contenido de la pila en orden inverso.

```
Stack s = new Stack();
// llenar la pila
for(int i = 0; i < 5; i++)
    s.push( i );
// imprimir el contenido de la pila
// mientras lo vaciamos (¿?)
for(int i = 0; i < s.size(); i++)
    System.out.print( s.pop() + " ");
// Cuál es la salida?
```

A 0 1 2 3 4

B 4 3 2 1 0

C 4 3 2

D 2 3 4

E No hay salida debido a un error de tiempo de ejecución.

Pilas mediante la clase Stack – Versión correcta

Escriba un método para imprimir el contenido de la pila en orden inverso.

```
Stack s = new Stack();
// llenar la pila
for(int i = 0; i < 5; i++)
      s.push( i );
// imprime el contenido de la pila
// mientras lo vaciamos
int limit = s.size();
for(int i = 0; i < limit; i++)
      System.out.print( s.pop() + " ");
//o
//    while( !s.isEmpty() )
//          System.out.println( s.pop() );
```

Limitaciones de pila/modismos

- Recuerde: no puede recorrer una pila de la forma habitual.

```
Stack<Integer> s = new Stack<Integer>();
  ...
for (int i = 0; i < s.size(); i++) {
      hacer algo con s.get(i);
}
```

- En su lugar, debe sacar los contenidos de la pila para verlos.
  - modismo común: Quitar cada elemento hasta que la pila esté vacía.

```
while (!s.isEmpty()) {
      hacer algo con s.pop();
}
```

¿Qué pasó con mi pila?

- Supongamos que se nos pide que escribamos un método `max` que acepte una pila de enteros y devuelva el entero más grande de la pila.
  - La siguiente solución es aparentemente correcta:

```
// Precondición: s.size() > 0
public static void max(Stack<Integer> s) {
    int maxValue = s.pop();

    while (!s.isEmpty()) {
        int next = s.pop();
        maxValue = Math.max(maxValue, next);
    }
    return maxValue;
}
```

  - El algoritmo es correcto, pero ¿qué tiene de malo el código?

¿Qué pasó con mi pila?

- El código destruye la pila al averiguar su respuesta.
  - Para solucionar esto, debe guardar y restaurar el contenido de la pila:

```
public static void max(Stack<Integer> s) {
    Stack<Integer> backup = new Stack<Integer>();
    int maxValue = s.pop();
    backup.push(maxValue);

    while (!s.isEmpty()) {
        int next = s.pop();
        backup.push(next);
        maxValue = Math.max(maxValue, next);
    }

    while (!backup.isEmpty()) {
        s.push(backup.pop());
    }
    return maxValue;
}
```

Ejercicio

- Considere un archivo de entrada de puntajes de exámenes en orden ABC inverso:

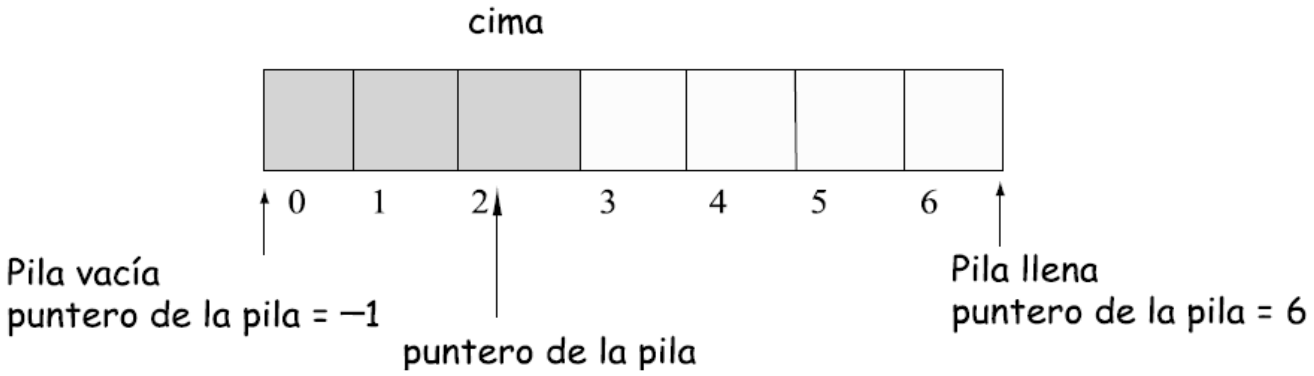
```
 Yeilding        Janet        87
White            Steven       84
Todd             Kim          52
Tashev           Sylvia       95
...
```

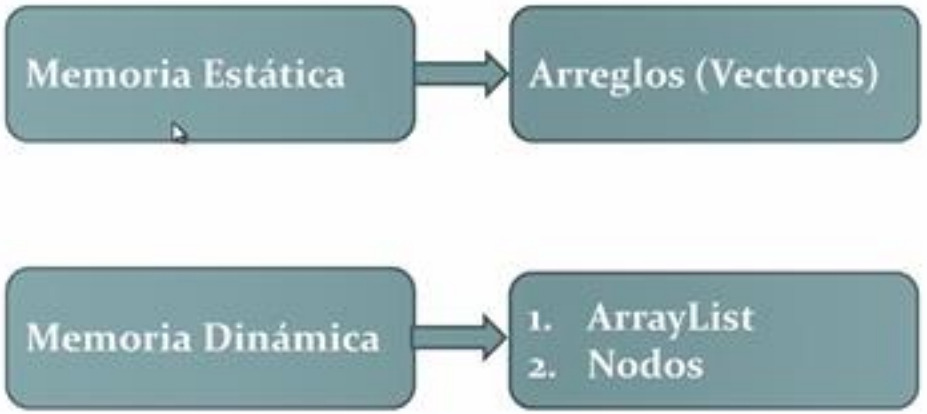
- Escriba código para imprimir los puntajes del examen en orden ABC usando una pila.

  – ¿Qué pasa si queremos seguir procesando los exámenes después de imprimirlos?

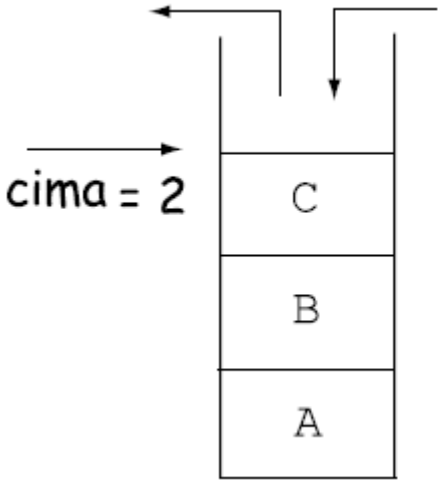
Representación de Pilas

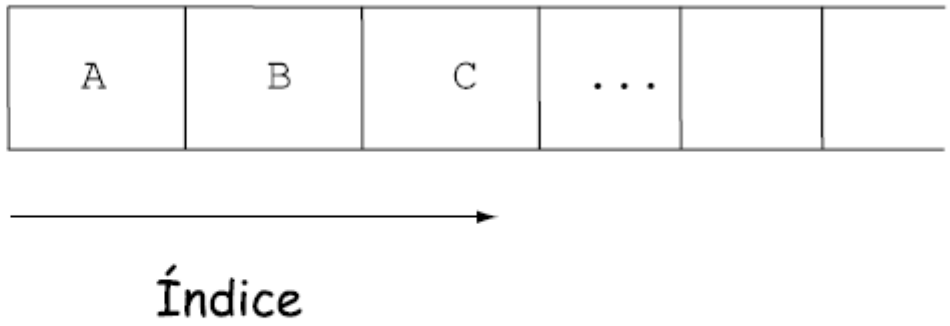
- Las pilas pueden representarse mediante el uso de :
  - Arreglos.
  - Listas enlazadas.
- **Limitación con arreglos:** Espacio de memoria reservada. Una vez establecido un máximo de capacidad para la pila, ya no es posible insertar más elementos:
  - Definir el tamaño máximo de la pila.
  - Un apuntador al último elemento insertado en la pila (cima)

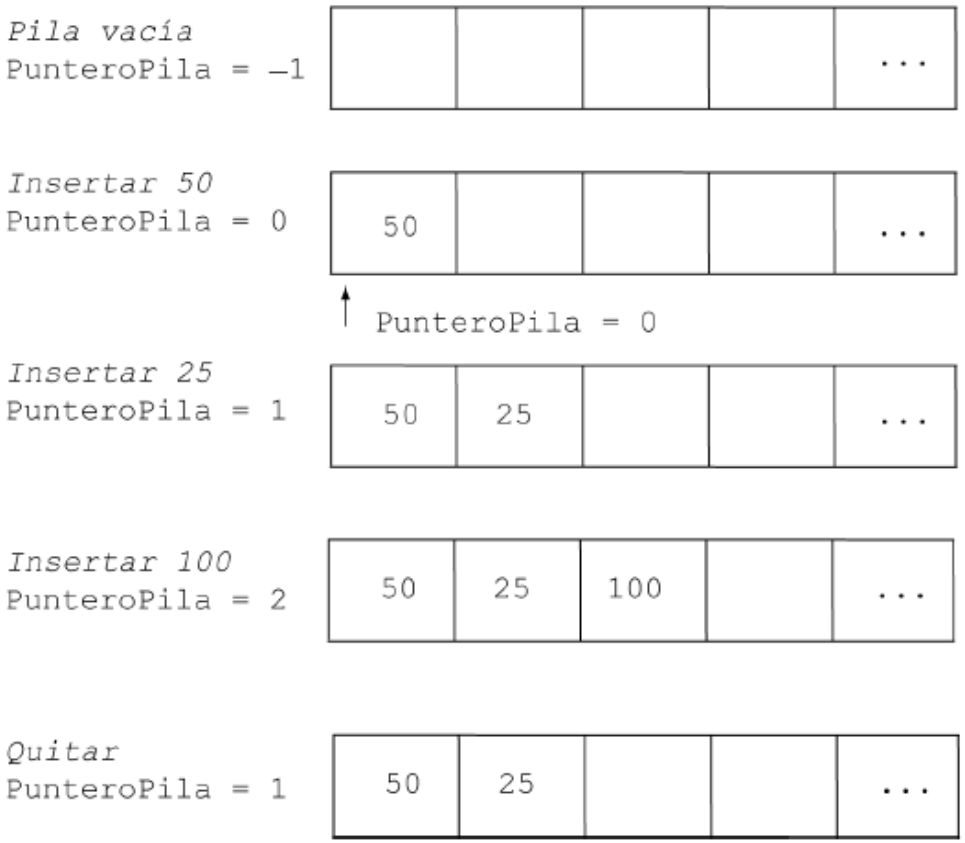




Representación de Pilas usando Arreglos



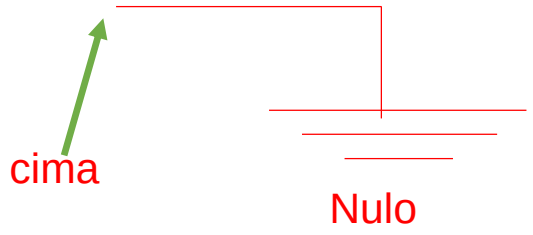


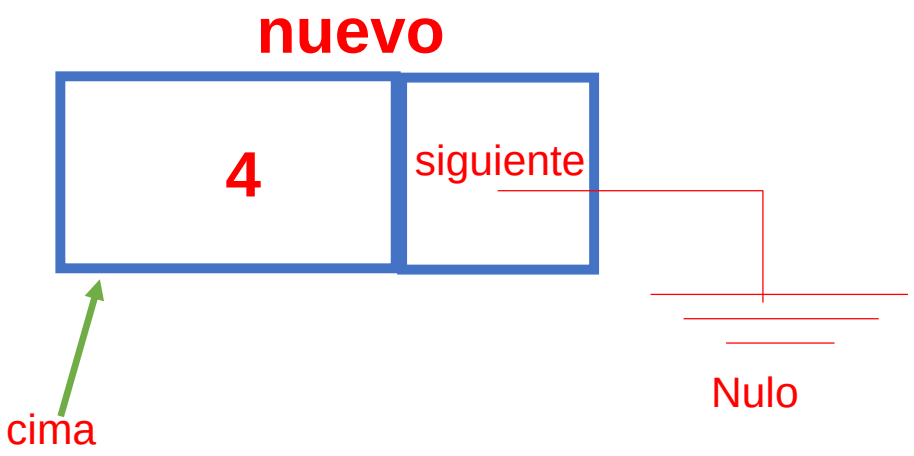


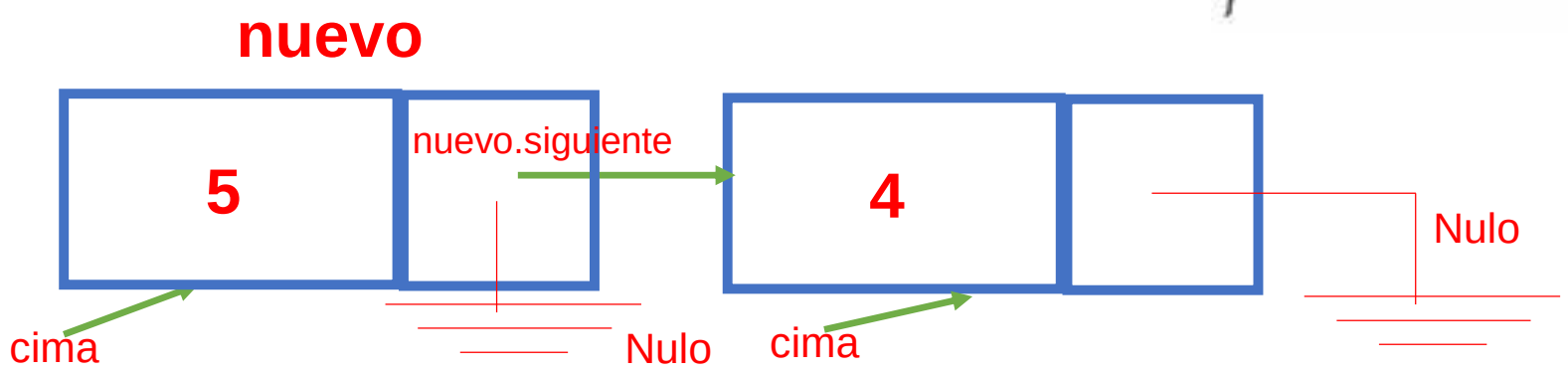
Implementación de una pila con listas enlazadas

```
public class NodoPila {
        int dato;
        NodoPila siguiente;
        public NodoPila(int d){
                dato=d;
                siguiente=null;
        }
}
```

Pilas con Lista Enlazada








```java
public class Pila {
    private NodoPila cima;
    int tama;
    public Pila(){
        cima=null;
        tama=0;
    }
    public boolean estaVacia(){
        return cima==null;
    }
    //Método para empujar (push) un elemento en la Pila
    public void empujar(int elem){
        NodoPila nuevo = new NodoPila(elem);
        nuevo.siguiente=cima;
        cima=nuevo;
        tama++;
    }
}
```

```java
public class NodoPila {
    int dato;
    NodoPila siguiente;
    public NodoPila(int d){
        dato=d;
        siguiente=null;
    }
}
```

Implementación de una pila con listas enlazadas

```
public class Pila {
    private NodoPila cima;
    int tama;
    public Pila(){
        cima=null;
        tama=0;
    }
    //Método para saber cuando la Pila está vacía
    public boolean estaVacia(){
        return cima==null;
    }
    //Metodo para insertar (push) un elemento en la p
    public void empujar(int elem){
        NodoPila nuevo = new NodoPila(elem);
        nuevo.siguiente=cima;
        cima=nuevo;
        tama++;
    }
    //Metodo para sacar(pop) un elemento en la pila
    public int sacar(){
        int auxiliar=cima.dato;
        cima=cima.siguiente;
        tama--;
        return auxiliar;
    }
    //Metodo para saber quien está en la cima de la pila
    public int cima(){
        return cima.dato;
    }
    //Método para saber el tamaño de la pila
    public int tamanioPila(){
        return tama;
    }
    //Metodo para Limpiar(Vaciar) la Pila
    public void limpiarPila(){
        while(!estaVacia()){
            sacar();
        }
    }
}
```

Implementación de una pila con listas enlazadas

```
package TipoPila;
import javax.swing.JOptionPane;
public class UsoPila {
    public static void main(String[] args) {
        int opcion = 0, elemento = 0;
        Pila pilon = new Pila();
        do {
            opcion = Integer.parseInt(JOptionPane.showInputDialog(null,
                    "1.Insertar un elemento en la pila\n"
                    + "2.Sacar un elemento de la pila\n"
                    + "3.¿La pila está vacia?\n"
                    + "4.¿Qué elemento está en la cima?\n"
                    + "5.Tamaño de la pila\n"
                    + "6.Vaciar Pila\n"
                    + "7.Salir", "Menu de Opciones de una Pila", JOptionPane.INFORMATION_MESSAGE));
```

Implementación de una pila con listas enlazadas

```
switch (opcion) {
    case 1:
        elemento = Integer.parseInt(JOptionPane.showInputDialog(null,
                "Ingresa el elemento a ingresar",
                "Insertando en la pila", JOptionPane.INFORMATION_MESSAGE));
        pilon.empujar(elemento);
        break;
    case 2:
        if (!pilon.estaVacia()) {
            JOptionPane.showMessageDialog(null, "El elemento obtenido es " + pilon.sacar()
                    "Obteniendo datos de la pila", JOptionPane.INFORMATION_MESSAGE);
        } else {
            JOptionPane.showMessageDialog(null, "La pila esta vacia",
                    "Pila vacia", JOptionPane.INFORMATION_MESSAGE);
        }
        break;
```

Implementación de una pila con listas enlazadas

```
case 3:
    if (pilon.estaVacia()) {
        JOptionPane.showMessageDialog(null, "La pila está vacia",
                "Pila vacia", JOptionPane.INFORMATION_MESSAGE);
    } else {
        JOptionPane.showMessageDialog(null, "La pila no esta vacia",
                "La Pila contiene datos", JOptionPane.INFORMATION_MESSAGE);
    }
    break;
case 4:
    if (!pilon.estaVacia()) {
        JOptionPane.showMessageDialog(null, "El elemento de la cima es: " + pilon.cima(),
                "Cima de la pila", JOptionPane.INFORMATION_MESSAGE);
    } else {
        JOptionPane.showMessageDialog(null, "La pila esta vacia",
                "Pila vacia", JOptionPane.INFORMATION_MESSAGE);
    }
    break;
```

Implementacion de una pila con listas enlazadas

```
case 5:
    JOptionPane.showMessageDialog(null, "El tamaño de la pila es: " + pilon.tamanioPila(),
            "Tamaño de la pila", JOptionPane.INFORMATION_MESSAGE);
    break;
case 6:
    if (!pilon.estaVacia()) {
        pilon.limpiarPila();
        JOptionPane.showMessageDialog(null, "La pila se ha limpiado",
                "Limpiando pila", JOptionPane.INFORMATION_MESSAGE);
    } else {
        JOptionPane.showMessageDialog(null, "La pila esta vacia",
                "Pila vacia", JOptionPane.INFORMATION_MESSAGE);
    }
    break;
case 7:
    JOptionPane.showMessageDialog(null, "Aplicacion Finalizada",
            "Fin", JOptionPane.INFORMATION_MESSAGE);
    break;
```

Implementacion de una pila con listas enlazadas

```
                default:
                    JOptionPane.showMessageDialog(null, "Opción Incorrecta",
                            "¡Cuidado!", JOptionPane.INFORMATION_MESSAGE);
            }
        } while (opcion != 7);

    }
}
```

Universidad Nacional Mayor de San Marcos
Universidad del Perú. Decana de América

Guía de Laboratorio

Colas

- ✓ Es una Estructura de Datos que almacena elementos en una Lista.
- ✓ Conocidas como FIFO.
- ✓ Los elementos se insertar por el final.
- ✓ Los elementos se eliminan por el inicio.

Operaciones Básicas de una cola

1. Crear Cola.
2. Insertar.
3. Quitar.
4. Cola Vacía.
5. Cola Llena.
6. Frente (Inicio).
7. Tamaño de la Cola.

Casos Típicos

Situación.	Llegadas.	Cola .	Mecanismo de Servicio.
Aeropuerto.	Aviones.	Aviones en vuelo.	Pista.
Aeropuerto.	Pasajeros.	Sala de espera.	Avión.
Depto. De bomberos.	Alarmas de incendio.	Incendios.	Depto. De Bomberos.
Compañía telefónica.	Números marcados.	Llamadas.	Conmutador.
Lavado de carros.	Autos.	Autos sucios.	Mecanismo de lavado.
La corte.	Casos.	Casos atrasados.	Juez.
Panadería.	Clientes.	Clientes con números.	Vendedor.
Carga de camiones.	Camiones.	Camiones en espera.	Muelle de carga.
Oficina de correos.	Cartas.	Buzón.	Empleados del correo.
Fábrica.	Ensamble.	Inventario en proceso.	Estación de trabajo.
Cartas de negocios.	Notas de dictado.	Cartas para mecanografiar.	Secretaria.
Producción.	Pedidos.	Trabajos.	Entrega del producto terminado.
Hospital.	Pacientes.	Personas enfermas.	Hospital.

Estructuras de datos lineales

- **Especificación formal:**

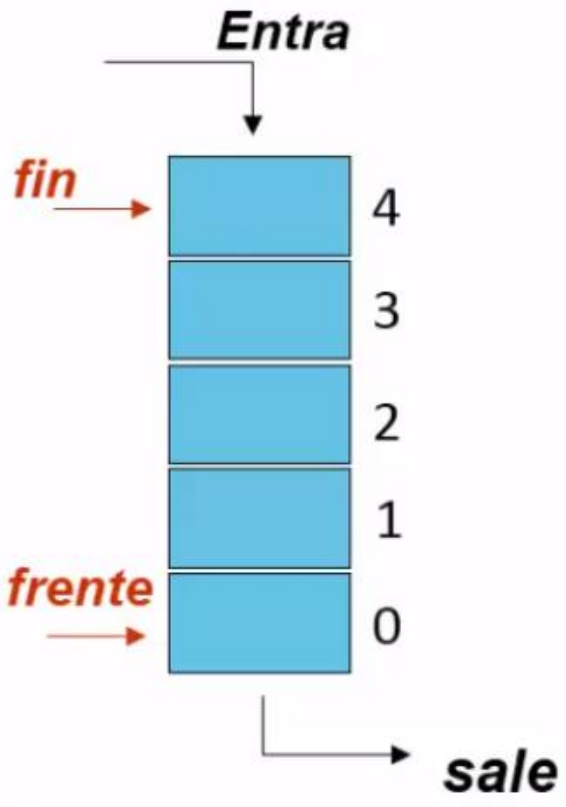
```
public interface IQueue {

    public boolean isEmpty();
    public void enqueue(String elem);
    public String dequeue();
    public String front();
    public int getSize();

}
```

En estas diapositivas, usamos una cola de objetos String, pero recuerda que puede definir una cola de cualquier tipo de datos

Colas (Filas)



Definición

Estructura de datos abstracto lineal cuyos elementos son insertados y eliminados de acuerdo con el principio de que el primero en entrar es el primero en salir **(FIFO)**

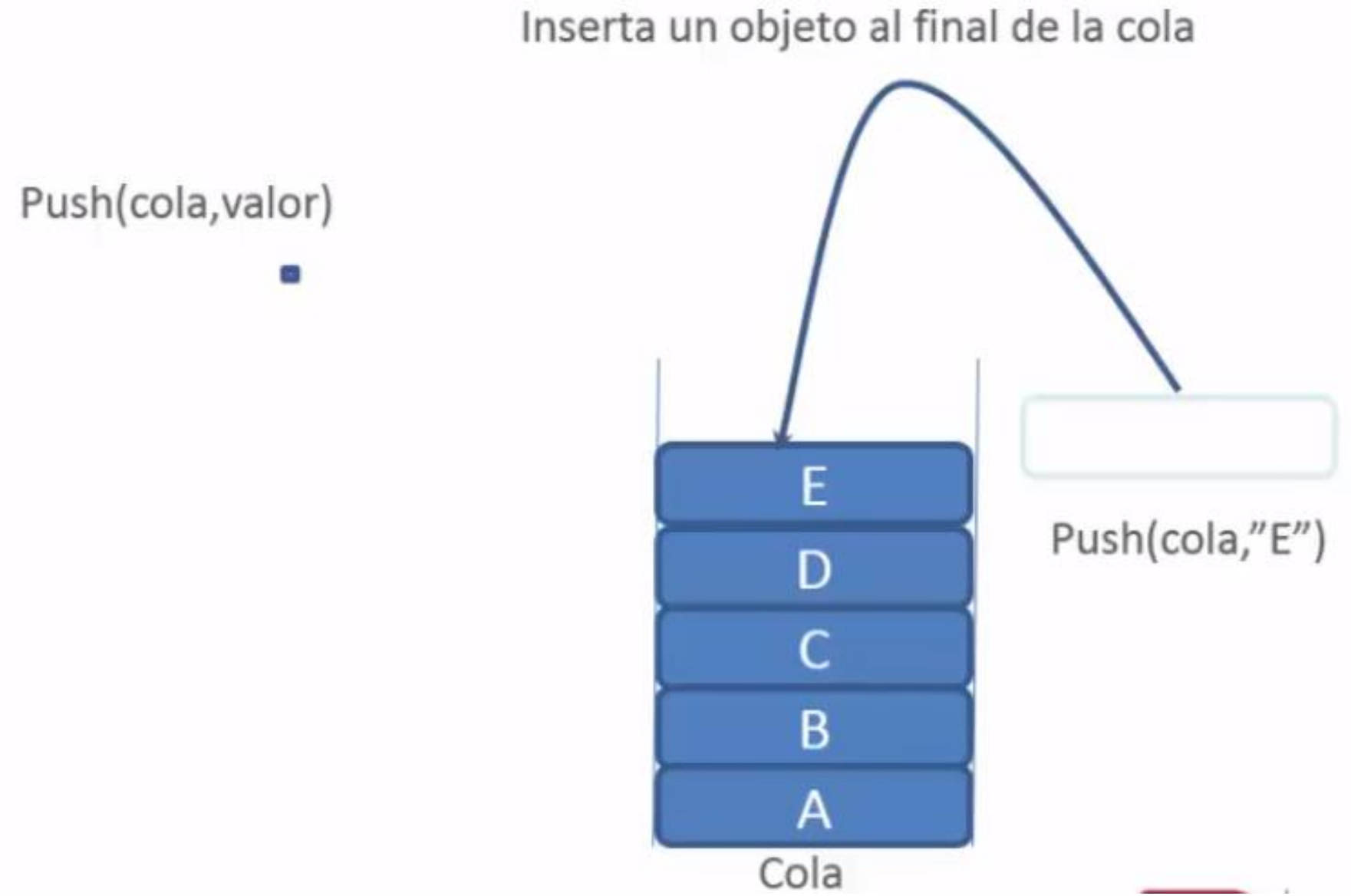
Una **cola** es un caso particular de lista en el cual los elementos se insertan en un extremo (el posterior o final) y se suprimen en el otro (el anterior o frente)

Un ejemplo de cola es la Cola de impresión en el sistema operativo Windows. Cada usuario de una red de Windows coloca sus trabajos de impresión y el sistema lo imprime en el mismo orden en que fueron insertados en la cola de impresión.

Operaciones con colas

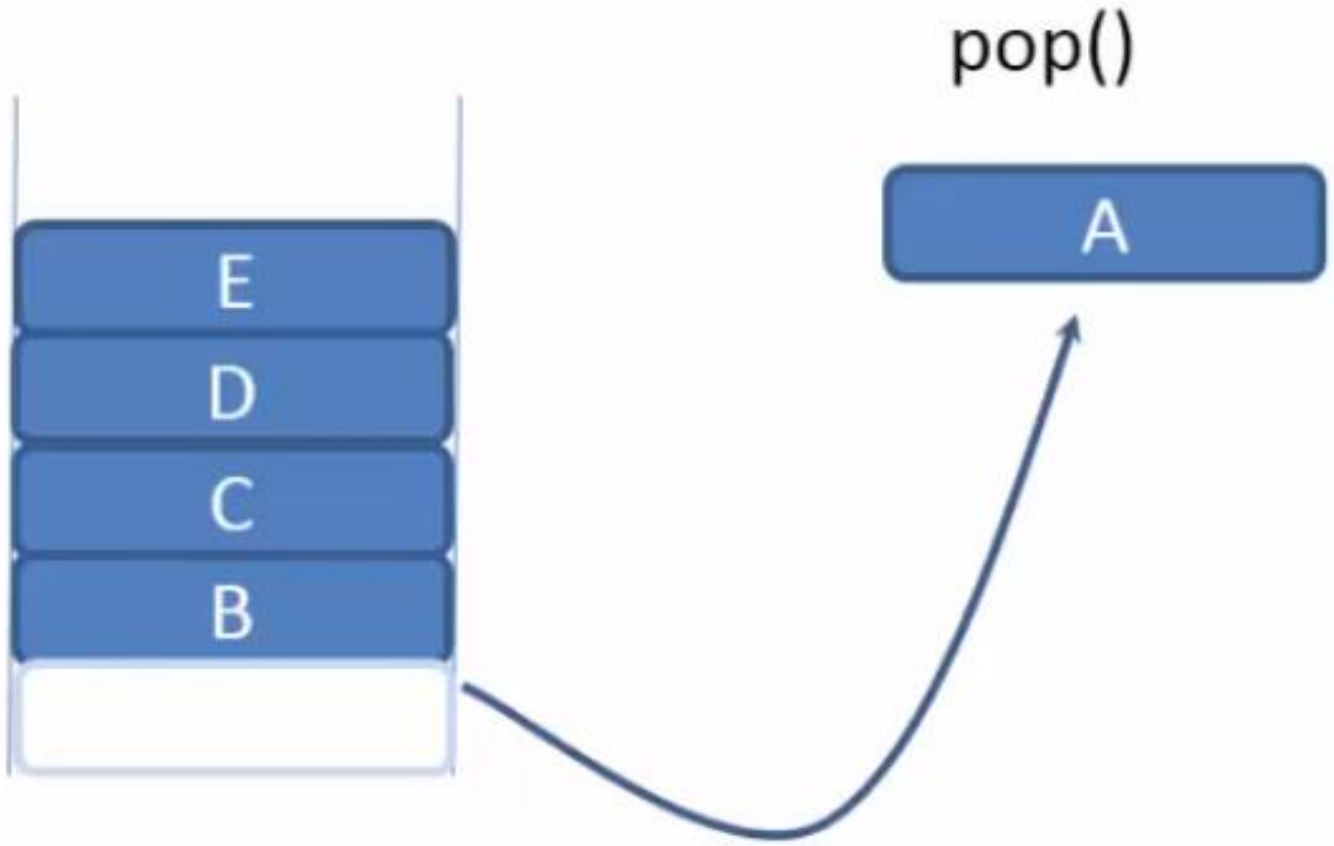
- ***push:*** agregar un elemento al final de la cola.
- ***pop :*** quitar un elemento del inicio de la cola.
- ***Empty:*** retorna un valor que indica si la cola está vacía o no.
- ***Top:*** retorna el objeto que está al inicio de la cola sin borrarlo.

Operaciones con Colas: Encolar (push)



Operaciones con Colas: Desencolar (pop)

Remueve el objeto que está al inicio de la cola



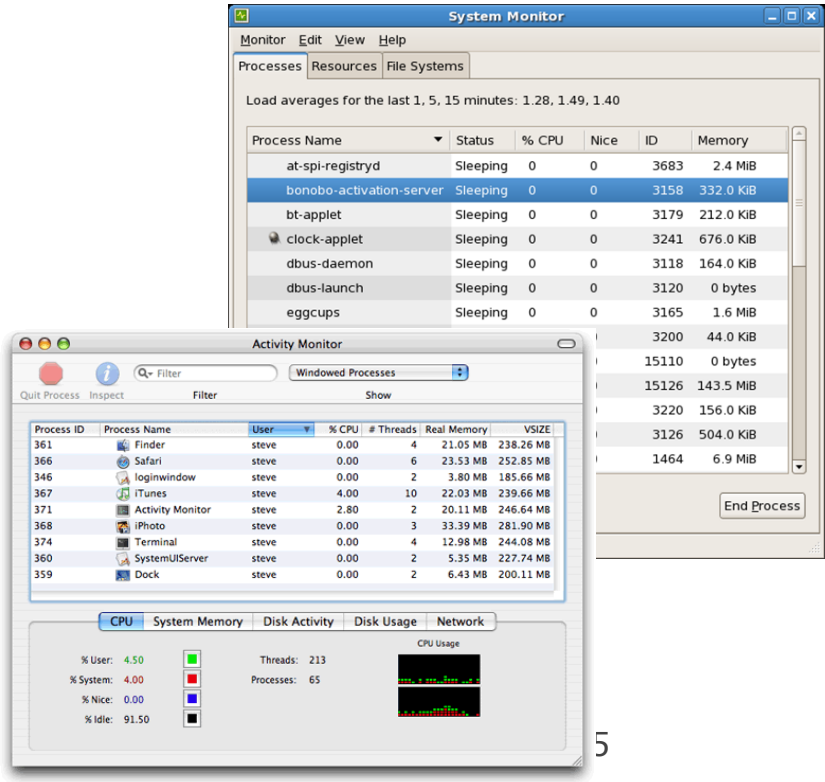
Colas en informática

- Sistemas operativos:
  - cola de trabajos de impresión para enviar a la impresora
  - cola de programas/procesos a ejecutar
  - cola de paquetes de datos de red para enviar

- Programación:
  - modelado de una línea de clientes o clientes
  - almacenar una cola de cálculos a realizar en orden

- Ejemplos del mundo real:
  - gente en una escalera mecánica o esperando en una fila
  - coches en una gasolinera (o en una cadena de montaje)

- En una computadora con N núcleos en la CPU, pero más de N procesos, ¿cuántos procesos se pueden ejecutar al mismo tiempo?
- Un trabajo de sistema operativo, programar los procesos para la CPU



Tipos de Cola

TIPOS DE COLAS:

- **Cola simple:** Estructura lineal donde los elementos salen en el mismo orden en que llegan.
- **Cola circular:** Representación lógica de una cola simple en un arreglo.
- **Cola de Prioridades:** Estructura lineal en la cual los elementos se insertan en cualquier posición de la cola y se remueven solamente por el frente.
- **Cola Doble (Bicola):** Estructura lineal en la que los elementos se pueden añadir o quitar por cualquier extremo de la cola (cola bidireccional).
  - **De entrada restringida**
  - **De salida restringida**

Operaciones Básicas en Colas Simples

Crear: se crea la cola vacía.

Insertar (Encolar, añadir, entrar, push).- Almacena al final de la cola el elemento que se recibe como parámetro.

Eliminar (Desencolar, sacar, salir, pop).- Saca de la cola el elemento que se encuentra al frente.

Vacía.- Regresa un **valor booleano** indicando si la cola ***tiene o no elementos*** (true – si la cola esta vacia, false – si la cola tiene al menos un elemento).

Llena.- Regresa un **valor booleano** indicando si la cola ***tiene espacio disponible*** para insertar nuevos elementos (**true** – si esta llena y **false** si existen espacios disponibles).

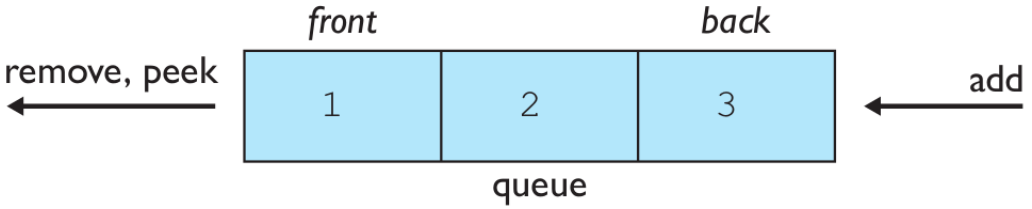
Frente (consultar, front).- se devuelve el elemento frontal de la cola, el primer elemento que entró.

Operaciones Básicas en Colas Simples

- Las operaciones básicas de una cola son :
  - **Enqueue** - meter: añade un nuevo elemento a final de la cola
  - **Dequeue** - sacar: elimina (saca) el primer elemento de la cola
- Otras operaciones usualmente incluidas en el tipo abstracto :
  - ***isEmpty*** (estáVacia): verifica si la cola está vacía
  - ***isFull*** (estáLlena): verifica si la cola está llena

Colas

- **queue** : recupera los elementos en el orden en que se agregaron.
  - Primero en entrar, primero en salir ("FIFO")
  - Los elementos se almacenan en orden de inserción pero no tienen índices.
  - El cliente solo puede agregar al final de la cola y solo puede examinar/eliminar el frente de la cola.



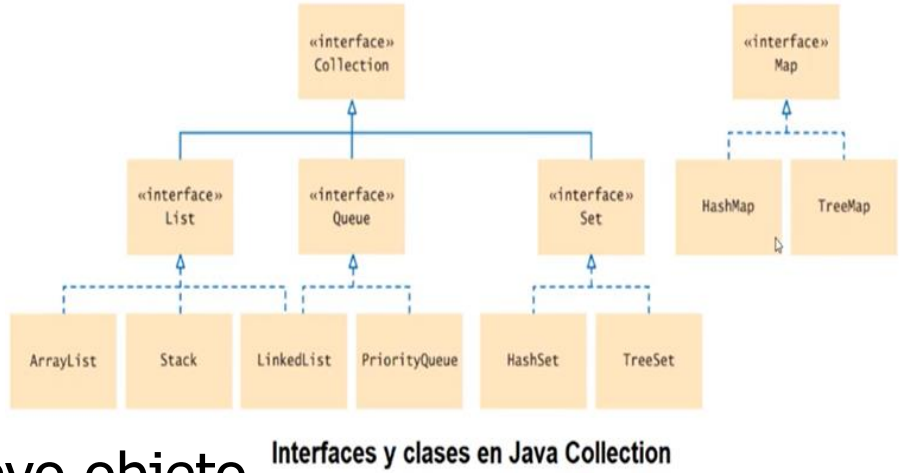
- operaciones básicas de cola:
  - **add** (enqueue): Agrega un elemento en la parte posterior.
  - **remove** (dequeue): Retira el elemento frontal.
  - **peek** : Examina el elemento frontal.

Programación con Queues

`add(`**valor**`)`	coloca el valor dado al final de la cola
`remove()`	elimina el valor del frente de la cola y lo devuelve; lanza una `NoSuchElementException` si la cola está vacía
`peek()`	devuelve el valor inicial de la cola sin eliminarlo; devuelve `null` si la cola está vacía
`size()`	devuelve el número de elementos en la cola
`isEmpty()`	devuelve `true` si la cola no tiene elementos

```
Queue<Integer> q = new LinkedList <Integer>();
q.add(42);
q.add(-3);
q.add(17); // frente [42, -3, 17] atrás

System.out.println(q.remove()); // 42
```



Interfaces y clases en Java Collection

– **IMPORTANTE** : al construir una cola, debe usar un nuevo objeto `LinkedList` en lugar de un nuevo objeto `Queue` .
  - Esto tiene que ver con *interfaces* .

Modismos de cola

- Al igual que con las pilas, debe sacar los contenidos de la cola para verlos.

```
while (!q.isEmpty()) {
        hacer algo con q.remove();
}
```

– otro modismo: examinar cada elemento exactamente una vez.

```
int size = q.size();
for (int i = 0; i < size; i++) {
        hacer algo con q.remove();
        (incluida la posibilidad de volver a agregarlo a la cola)
}
```

- ¿Por qué necesitamos la variable `size` ?

Mezclando pilas y colas

- A menudo mezclamos pilas y colas para lograr ciertos efectos.
  - Ejemplo: Invertir el orden de los elementos de una cola.

```
Queue<Integer> q = new LinkedList<Integer>();
q.add(1);
q.add(2);
q.add(3); // [1, 2, 3]
Stack<Integer> s = new Stack<Integer>();
while (!q.isEmpty()) { // Q -> S
s.push(q.remove());
}
while (!s.isEmpty()) { // S -> Q
q.add(s.pop());
}

Sistema.out.println(q); // [3, 2, 1]
```

Ejercicio

- Modifique nuestro programa de puntuación de exámenes para que lea las puntuaciones de los exámenes en una cola e imprima la cola.
  - A continuación, filtre los exámenes en los que el alumno obtuvo una puntuación de 100.
  - Luego realice su código anterior de invertir e imprimir a los estudiantes restantes.
    - ¿Qué pasa si queremos seguir procesando los exámenes después de imprimirlos?

Ejercicios

- Escriba un método `stutter` que acepte una cola de enteros como parámetro y reemplace cada elemento de la cola con dos copias de ese elemento.

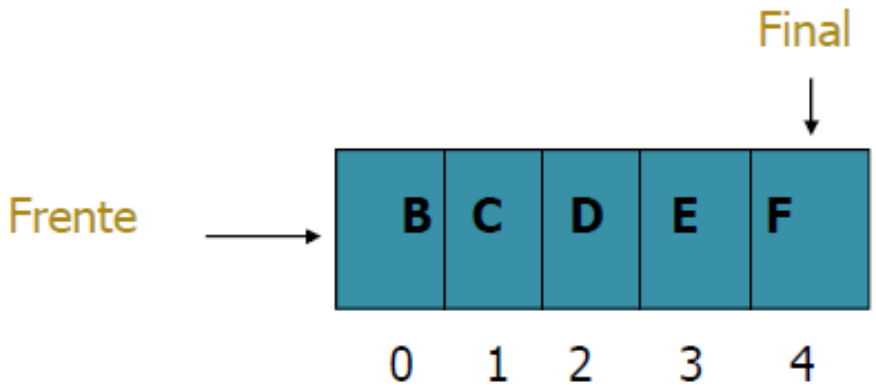
  - `frente [1, 2, 3] atrás`
    se convierte en
    `frente [1, 1, 2, 2, 3, 3] atrás`

- Escriba un método `mirror` que acepte una cola de cadenas como parámetro y agregue el contenido de la cola a sí mismo en orden inverso.

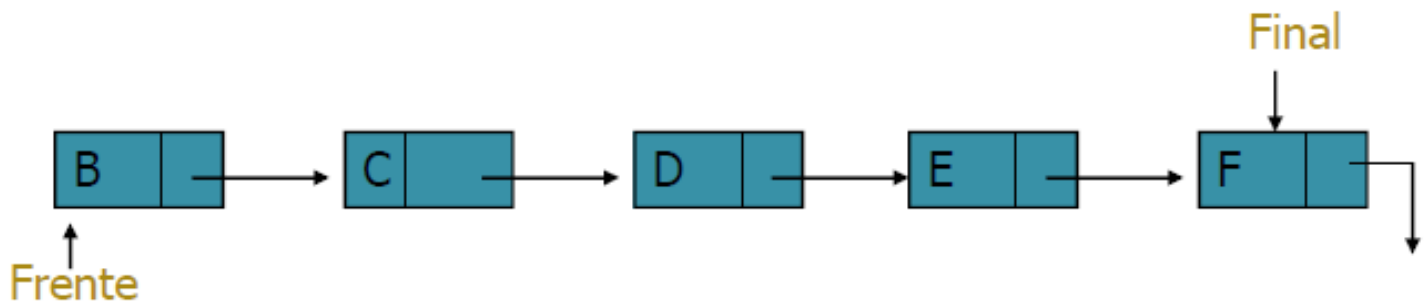
  - `frente [a, b, c] atrás`
    se convierte en
    `frente [a, b, c, c, b, a] atrás`

Representación de colas:

- Usando memoria estática: arreglos con tamaño fijo y frente fijo o movible o represntación circular.



- Usando memoria dinámica: Listas ligadas.



Operaciones con Colas

Operaciones:

1.- Insertar A

2.- Insertar B

3.- Insertar C

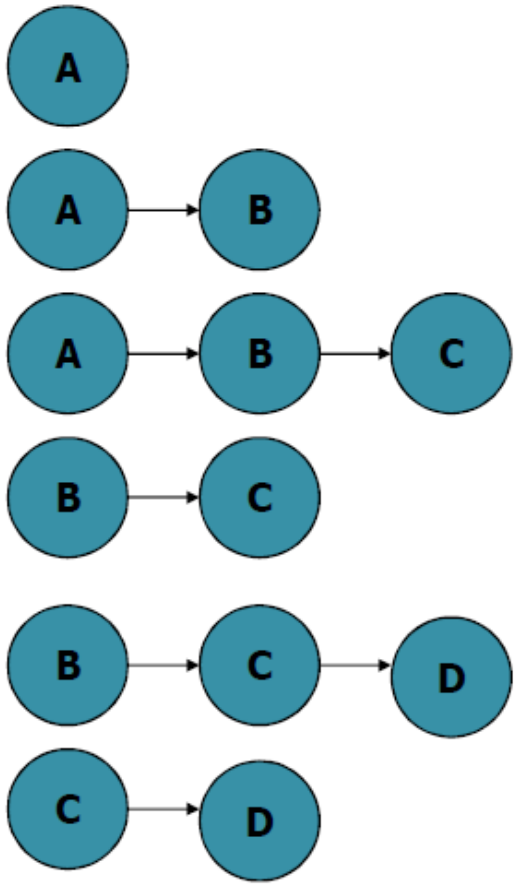
4.- Remover Elemento

5.- Insertar D

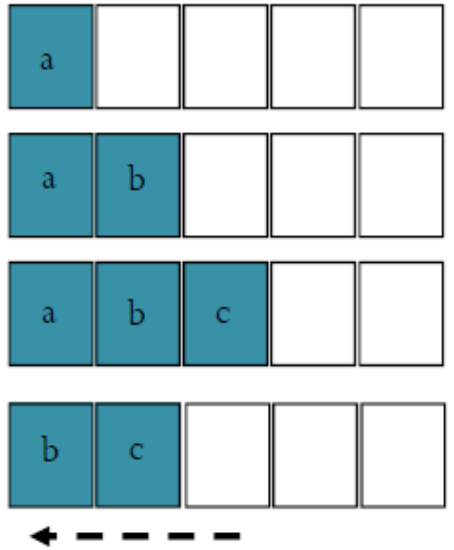
6.- Remover Elemento

Estado de la cola:

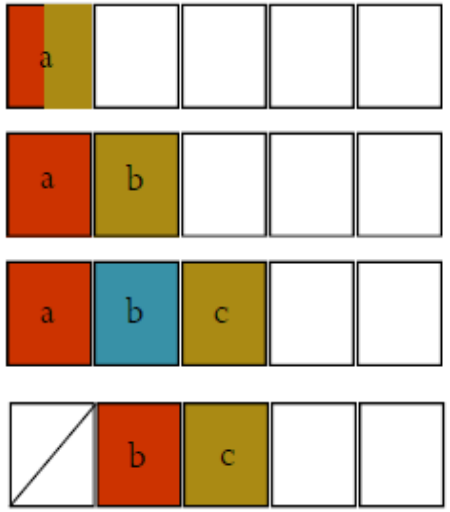
Inicio: Cola Vacía



Ejemplo



No es necesario mover todos los elementos



Apuntadores al frente y a la cola

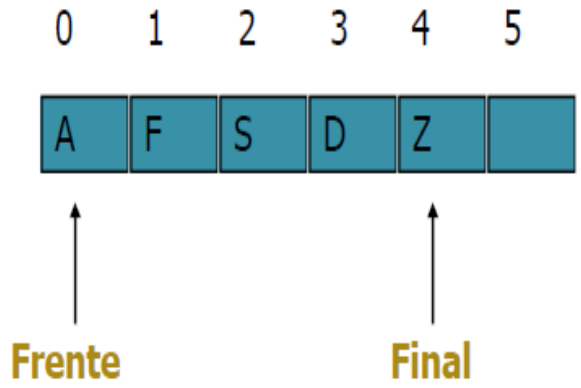
Representación de Colas usando arreglos

ColaArray
-MAX:int -Cola:int[] -frente:int -fin:int
+ColaArray(int) +isColaVacia():boolean +isColaLlena():boolean +insertarCola(int) +eliminarCola():int +vaciarCola() +mostrarCola() +primeroCola():int +tamaCola():int

Representación de Colas usando Arreglos

Representación usando arreglos

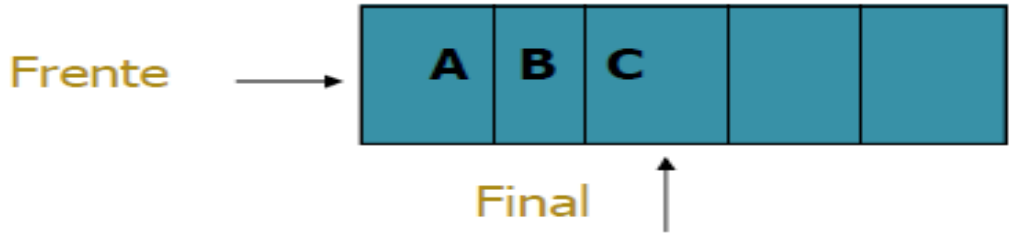
Las colas pueden ser representadas en arreglos de una dimensión (vector) manteniendo dos variables que indiquen el FRENTE y FINAL de los elementos de la cola.



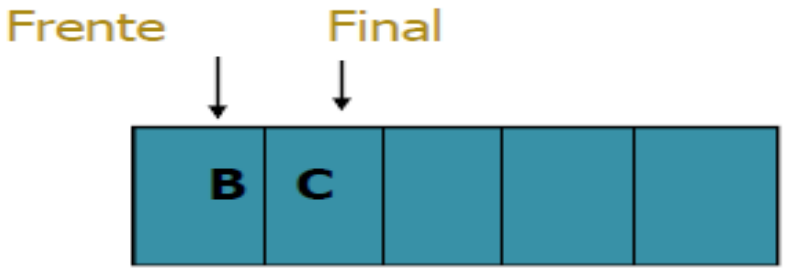
- Cuando la **cola esta vacía** las variables **frente** y **final** son nulos y no es posible remover elementos.
- Cuando la **cola esta llena** (**frente** = 0 y **final** = n-1) no es posible insertar elementos nuevos a la cola.
- Cuando se **remueven elementos** el **frente** puede incrementarse para apuntar al siguiente elemento de la cola (***implementación con frente móvil***) o los elementos en la cola pueden desplazarse una posición adelante (***implementación con frente fijo***)
- **Recuperación de espacio**: Cuando no hay espacios libres al final del arreglo los elementos pueden ser desplazados para desocupar posiciones en un extremo del arreglo o se puede manejar una estructura circular.

Representación de Colas usando Arreglos

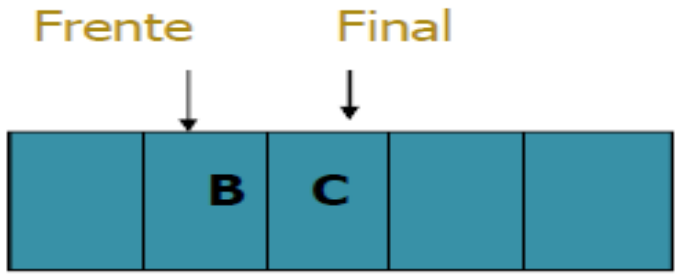
Ejemplo: Suponer que usamos un arreglo de 5 posiciones.
Usando la representación de frente fijo y frente movible.



Al remover un elemento:



Frente fijo

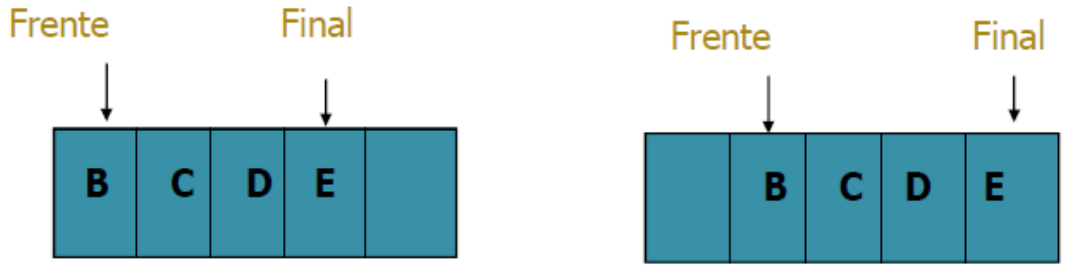


Frente movible

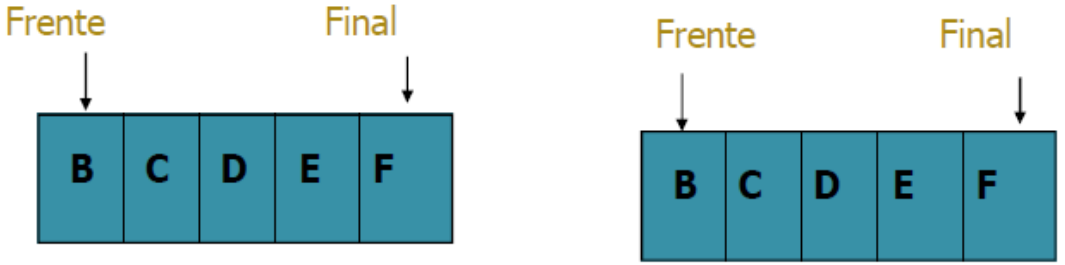
Representación de Colas usando Arreglos

Insertar elemento D:

Insertar elemento E:



Insertar elemento F:

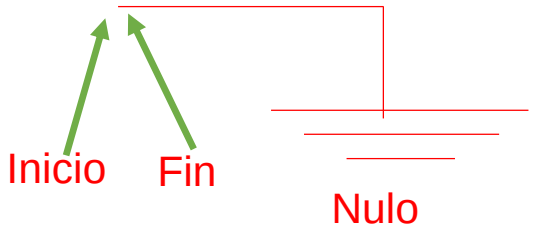


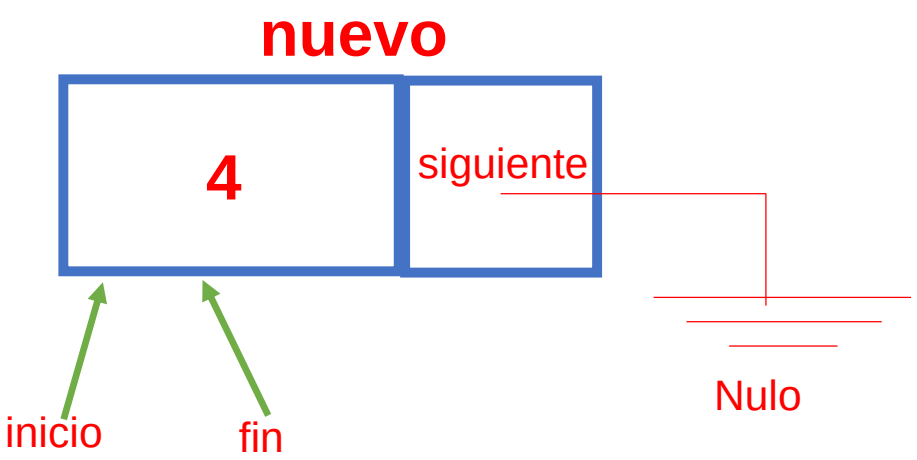
Insertar elemento G: Error: Cola llena!!!!

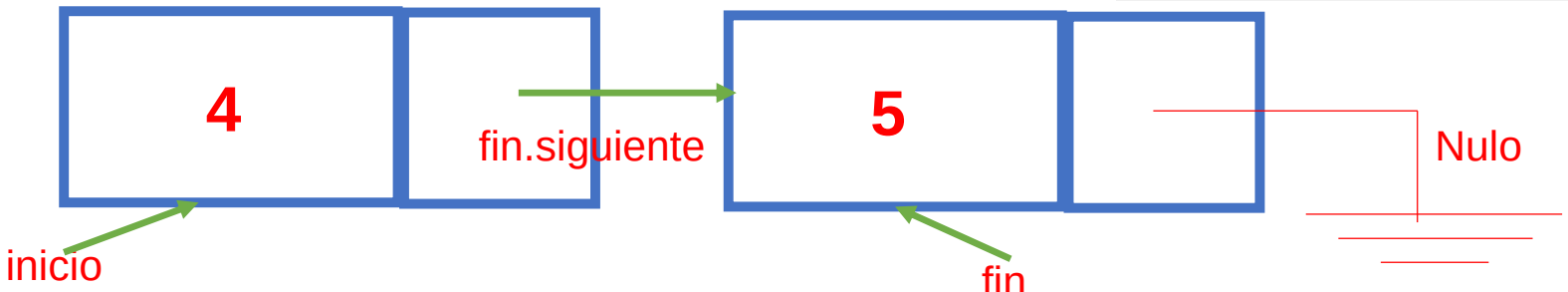
Implementación de una cola con listas enlazadas

```
public class Nodo {
   public int dato;
   public Nodo siguiente;//puntero enlace
//constructor para agregar un nodo al final
   public Nodo(int d)
   {
      this.dato = d;
      this.siguiente = null;
   }
}
```

Cola con Lista Enlazada








```java
public class ColaLista {
    //Toda lista debe referenciar al inicio y al fin
    protected Nodo inicio, fin;//punteros para saber donde está el inicio y el fin
    int tama;
    //Constructor crea una cola vacia
    public ColaLista(){
        inicio=null;
        fin=null;
        tama=0;
    }
    //método para verificar que la cola está vacía
    public boolean colaVacia()
    {
      return (inicio==null);
    }
    //Método para agregar un nodo al final
    public void insertar(int elemento)
    {
      //Creando al nodo
      Nodo nuevo=new Nodo(elemento);
      if (colaVacia())
      {
        inicio=nuevo;
      }
      else
      {
        fin.siguiente=nuevo;
      }
      fin=nuevo;
      tama++;
    }
}
```

```java
public class Nodo {
  public int dato;
  public Nodo siguiente;//puntero enlace
//constructor para agregar un nodo al final
  public Nodo(int d)
  {
    this.dato = d;
    this.siguiente = null;
  }
}
```

Implementación de una cola con listas enlazadas

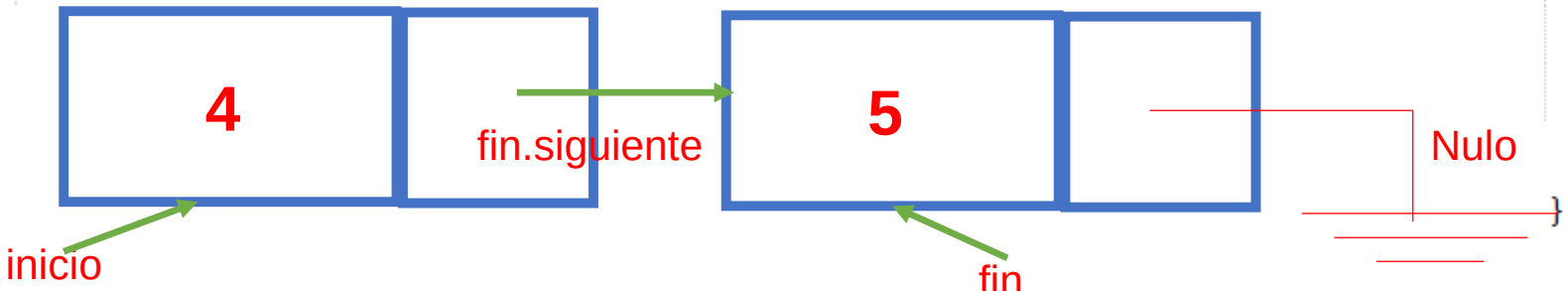
```java
package TipoCola;
public class ColaLista {
    //Toda lista debe referenciar al inicio y al fin
    protected Nodo inicio, fin;//punteros para saber donde está el inicio y el fin
    int tama;
    //Constructor crea una cola vacia
    public ColaLista(){
        inicio=null;
        fin=null;
        tama=0;
    }
    //método para verificar que la cola está vacía
    public boolean colaVacia()
    {
      return (inicio==null);
    }
```

```java
//Método para agregar un nodo al final
public void insertar(int elemento)
{
  //Creando al nodo
  Nodo nuevo=new Nodo(elemento);
  if (colaVacia())
  {
    inicio=nuevo;
  }
  else
  {
    fin.siguiente=nuevo;
  }
  fin=nuevo;
  tama++;
}
```

Implementacion de una cola con listas enlazadas

```
package TipoCola;
public class ColaLista {
    //Toda lista debe referenciar al inicio y al fin
    protected Nodo inicio, fin;//punteros para saber donde está el inicio y el fin
    int tama;
    //Constructor crea una cola vacia
    public ColaLista(){
        inicio=null;
        fin=null;
        tama=0;
    }
    //método para verificar que la cola está vacía
    public boolean colaVacia()
    {
      return (inicio==null);
    }
```

```
    public int quitar()
    {
      int aux;
        aux = inicio.dato;
        inicio = inicio.siguiente;
        tama--;
        return aux;
    }
    public int inicioCola(){
        return inicio.dato;
    }
    public int tamaCola(){
        return tama;
    }
}
```



Implementación de una cola con listas enlazadas

```
package TipoCola;
import javax.swing.JOptionPane;
public class UsoCola {
    public static void main(String[] args) {
        int opcion=0,elemento=0;
        ColaLista colon=new ColaLista();
        do{
            opcion=Integer.parseInt(JOptionPane.showInputDialog(null,
                    "1.Insertar un elemento en la cola\n"+
                            "2.Quitar un elemento de la cola\n"+
                            "3.¿La cola está vacia?\n"+
                            "4.Elemento ubicado al inicio de la cola\n"+
                            "5.Tamaño de la cola\n"+
                            "6.Salir","Menu de Opciones de una Cola",JOptionPane.QUESTION_MESSAGE));
```

Implementación de una cola con listas enlazadas

```
switch(opcion){
    case 1:
        elemento=Integer.parseInt(JOptionPane.showInputDialog(null,
        "Ingresa el elemento a ingresar",
        "Insertando en la cola",JOptionPane.QUESTION_MESSAGE));
        colon.insertar(elemento);
        break;
    case 2:
        if(!colon.colaVacia()){
            JOptionPane.showMessageDialog(null,"El elemento atendido es "+colon.quitar(),
                    "Quitando elementos de la cola",JOptionPane.INFORMATION_MESSAGE);
        }else{
            JOptionPane.showMessageDialog(null,"La cola esta vacia",
                    "Cola vacia",JOptionPane.INFORMATION_MESSAGE);
        }
        break;
```

Implementación de una cola con listas enlazadas

```
case 3:
    if(colon.colaVacia()){
        JOptionPane.showMessageDialog(null,"La cola está vacia",
                "Cola vacia",JOptionPane.INFORMATION_MESSAGE);
    }else{
        JOptionPane.showMessageDialog(null,"La cola no esta vacia",
                "Cola no vacia",JOptionPane.INFORMATION_MESSAGE);
    }
    break;
case 4:
    if(!colon.colaVacia()){
        JOptionPane.showMessageDialog(null,"El elemento ubicado al inicio de la cola es "+colon.inicioCola()
                "Mostrando el inicio de la cola",JOptionPane.INFORMATION_MESSAGE);
    }else{
        JOptionPane.showMessageDialog(null,"La cola esta vacia",
                "Cola vacia",JOptionPane.INFORMATION_MESSAGE);
    }
    break;
```

Implementación de una cola con listas enlazadas

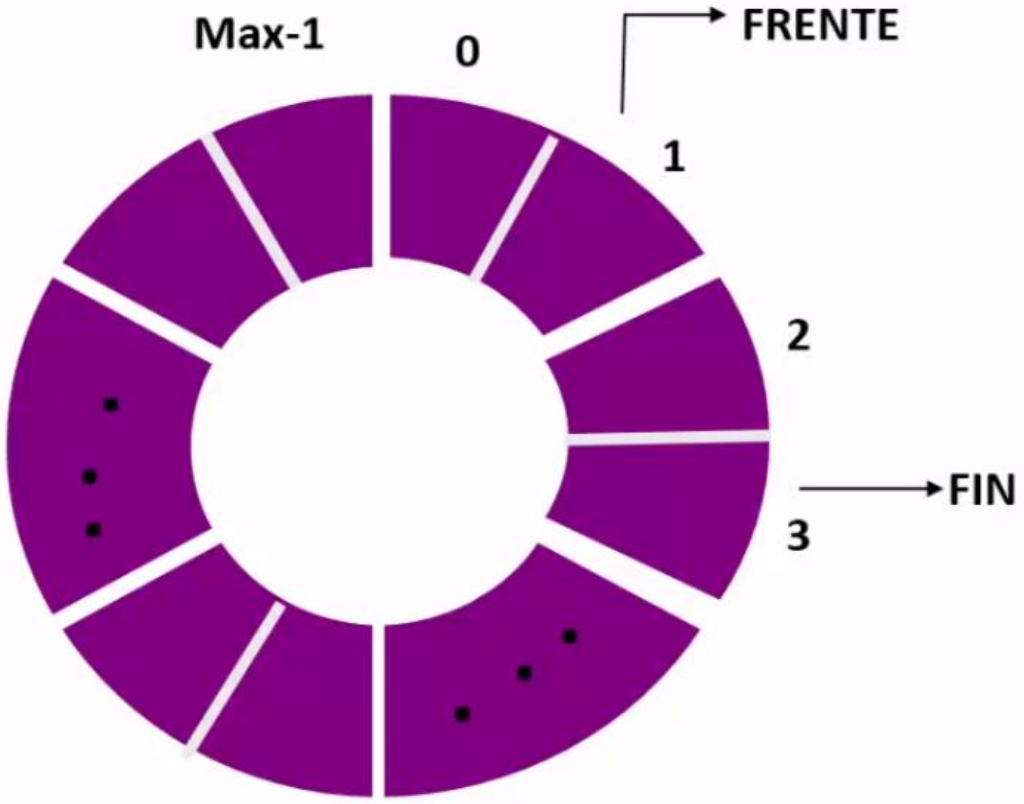
```
                case 5:
                    JOptionPane.showMessageDialog(null,"El tamaño de la cola es "+colon.tamaCola(),
                        "Mostrando el tamaño de la cola",JOptionPane.INFORMATION_MESSAGE);
                    break;
                case 6:
                    JOptionPane.showMessageDialog(null,"Aplicacion Finalizada",
                            "Fin",JOptionPane.INFORMATION_MESSAGE);
                    break;
                default:
                    JOptionPane.showMessageDialog(null,"Opción Incorrecta",
                            "¡Cuidado!",JOptionPane.INFORMATION_MESSAGE);
            }
        }while(opcion!=6);
    }
}
```

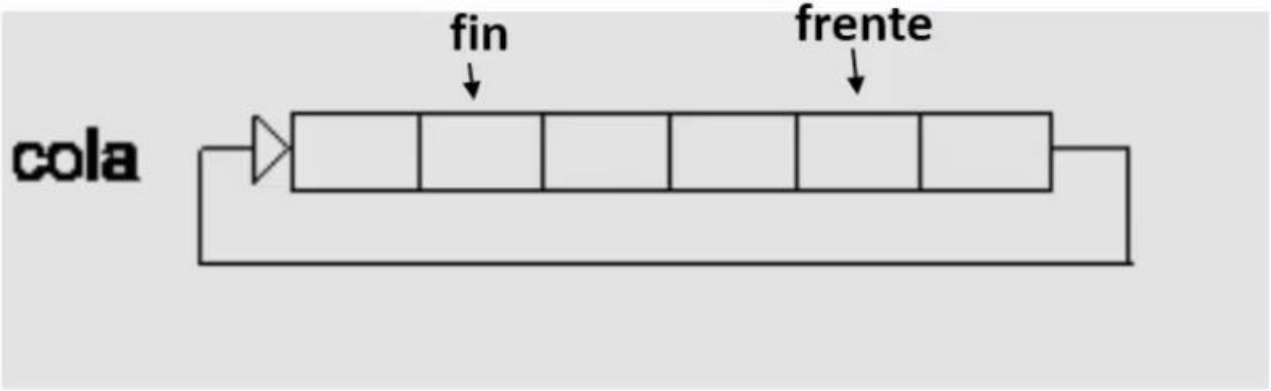
Colas circulares

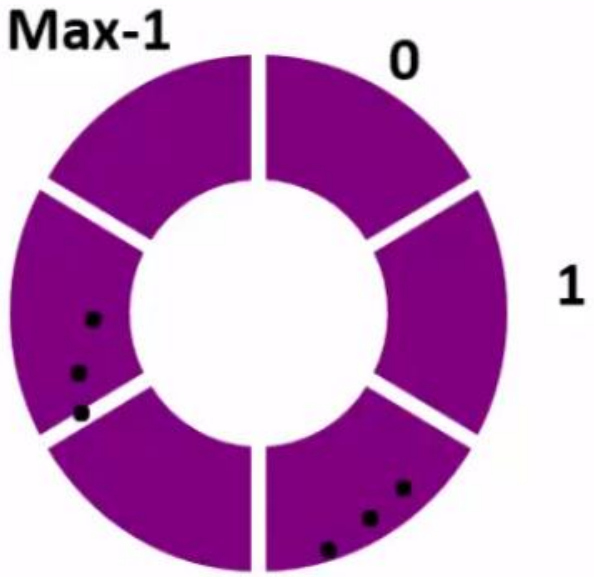
Colas circulares

Problema con las colas lineales al implementarlas con arreglos hacen que surjan las colas circulares.

La idea general es insertar elementos en posiciones previamente desocupadas







Colas circulares

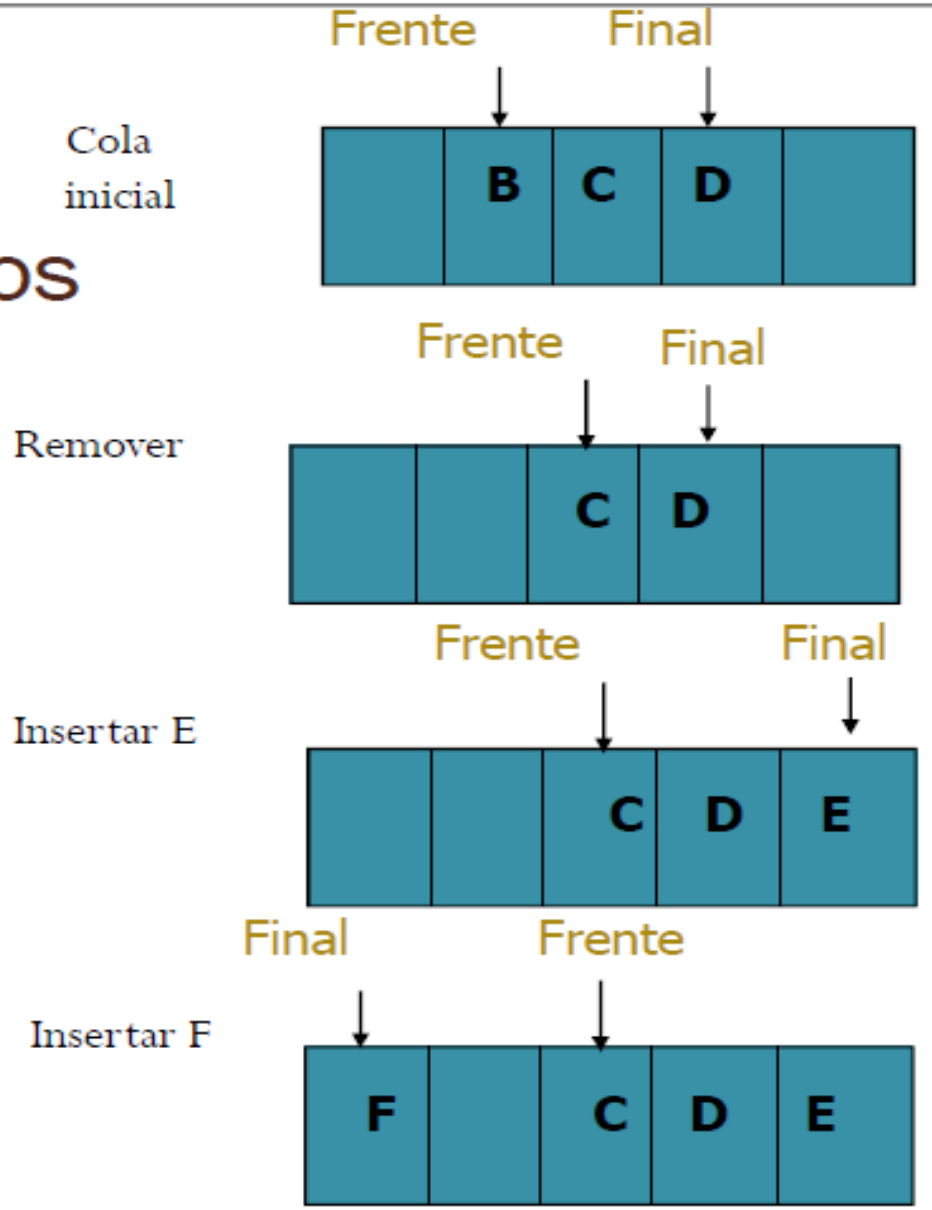
Cola Circular - Anillos

Es una representación lógica de la cola en un arreglo.

El **frente** y **final** son movibles.

Cuando el **frente** o **final** llegan al extremo se regresan a la primera posición del arreglo.

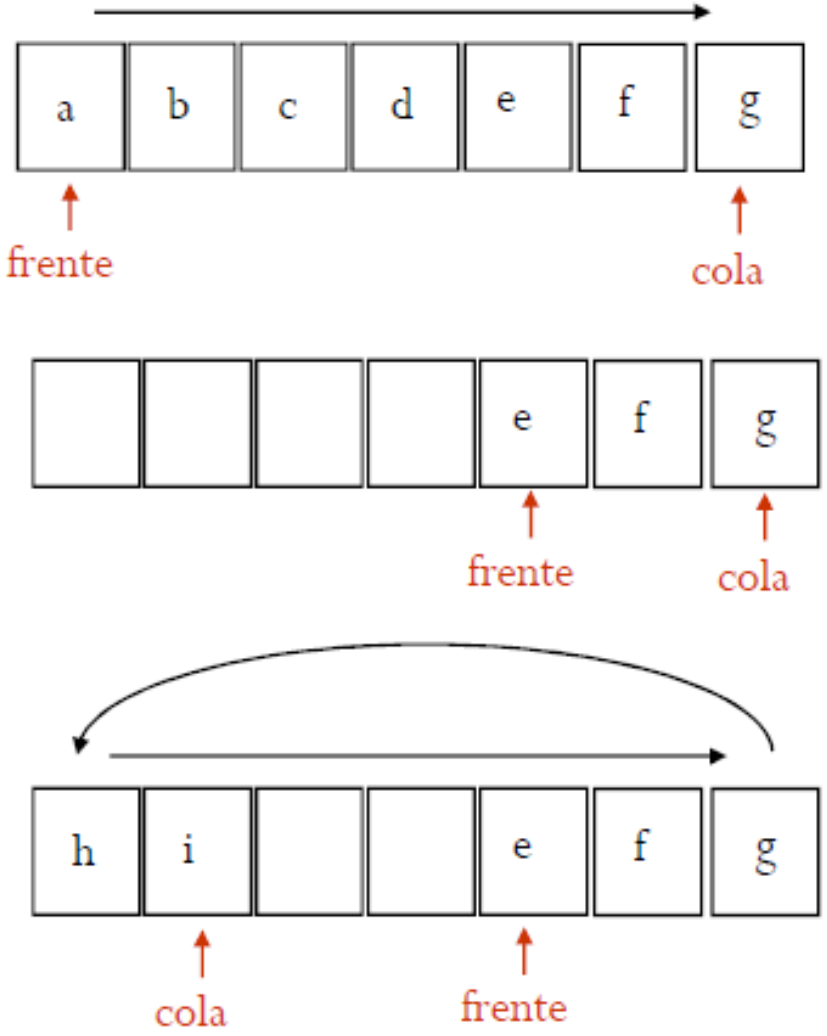
El último elemento y el primero están unidos.



Colas circulares

Colas circulares

- El objetivo de una cola circular es aprovechar al máximo el espacio del arreglo.
- La idea es insertar elementos en las localidades previamente desocupadas.
- La implementación tradicional considera dejar un espacio entre el frente y la cola.



Universidad Nacional Mayor de San Marcos
Universidad del Perú. Decana de América

Guía de Laboratorio

¿Preguntas?

Resumiendo y Repasando…

- Una pila es una colección que le permite agregar y quitar elementos de su parte superior, proporcionando acceso de "último en entrar, primero en salir" (LIFO).
- Las operaciones comunes de una pila incluyen agregar (“push"), eliminar ("pop"), probar si la pila está vacía, preguntar por el tamaño de la pila y "mirar" el elemento superior sin quitarlo (peek).
- Una cola es una colección que le permite agregar elementos en la parte posterior y eliminar elementos en la parte delantera, lo que proporciona acceso "primero en entrar, primero en salir" (FIFO).
- Las operaciones comunes de una cola incluyen agregar (“enqueue"), eliminar (“dequeue”), probar si la cola está vacía, preguntar por el tamaño de la cola y "mirar" el elemento frontal sin eliminarlo (peek).

Resumiendo y Repasando…

- Para procesar todos los elementos de una pila, se debe vaciar la colección. Si desea examinar los contenidos sin dañar la colección, debe mantener una copia de seguridad y restaurar los datos después.
- Para procesar todos los elementos de una cola, debe realizar una copia de seguridad y restaurar los datos o realizar un ciclo de valores hasta el final de la cola a medida que los procesa.
- El tamaño de una pila o cola cambia a medida que se procesan y eliminan sus elementos, por lo que muchos algoritmos para procesar estas colecciones deben realizar un seguimiento del tamaño de la colección por separado para evitar errores comunes.

FIN

Agenda

- Pilas y Colas
- Operaciones en las Pilas y Colas
- Estudio de caso: Evaluador de expresiones
- Implementando un evaluador de expresiones

Cálculos Matemáticos

¿Cuál es el resultado de 3 + 2 * 4 ? 2 * 4 + 3? si se evalúa de izquierda a derecha? y 5+5*2+5?

- La precedencia de los operadores afecta el orden de las operaciones. Una expresión matemática no se puede evaluar simplemente de izquierda a derecha.
- Ello es un desafío al diseñar un programa o algoritmo que evalúe estas expresiones.

Y cuál es el resultado de:

1 - 2 - 4 ^ 5 * 3 * 6 / 7 ^ 2 ^ 3

```
Paréntesis                : ( )
Potencia                  :  ^
Multiplicación/división   :  *, /
Suma/Resta                :  +, -
```

```
r =  a*b/(a+c)      g = a*b/a+c

r =  (a-b)^c+d      g = a-b^c+d
```

Expresiones Infija, Postfija y Prefija

- La forma en que estamos acostumbrados a escribir expresiones se conoce como notación infija
- La expresión postfija/prefija no require ninguna regla de precedencia
- 3 2 * 1 + es postfija(polaca inversa) de 3 * 2 + 1
- +*321 es es prefija de 3*2 + 1

Notación	Descripción	Ejemplos
Infija	Operador entre operandos	2.3 + 4.7 2.6 * 3.7 (3.4 + 7.9) * 18.6 + 2.3 / 4.7
Prefija	Operador antes de operandos (notación funcional)	+ 2.3 4.7 * 2.6 3.7 + * + 3.4 7.9 18.6 / 2.3 4.7
Postfija	Operador después de operandos (notación polaca inversa)	2.3 4.7 + 2.6 3.7 * 3.4 7.9 + 18.6 * 2.3 4.7 / +

```
a*b/(a+c)    (infija) → a*b/ac+  → ab*/ac+  → ab*ac+/  (polaca inversa)
a*b/a+c      (infija) → ab*/a+c  → ab*a/+c  → ab*a/c+  (polaca inversa)
(a-b)^c+d  (infija) → ab-^c+d → ab-c^+d → ab-c^d+ (polaca inversa)
```

Es importante tener en cuenta que en la notación postfija/prefija no son necesarios los paréntesis para cambiar el orden de evaluación.

Evaluación de una expresión aritmética

- La evaluación de una expresión aritmética escrita de manera habitual, en notación infija, se realiza en dos pasos principales:

1. Transformar la expresión de notación infija a postfija o prefija.

2. Evaluar la expresión en notación postfija o prefija.

Paso 1 - Ejemplo Simple

Expresión infija: 3 + 2 * 4

Expresión postfija:

Pila de operadores:

Tabla de prioridad

Simbolo	Off Stack Prioridad	On Stack Prioridad
+	1	1
-	1	1
*	2	2
/	2	2
^	4	3
(	5	0

Ejemplo Simple

Expresión infija: + 2 * 4
Expresión postfija: 3
Pila de operadores:

Tabla de prioridad

Simbolo	Off Stack Prioridad	On Stack Prioridad
+	1	1
-	1	1
*	2	2
/	2	2
^	4	3
(	5	0

Ejemplo Simple

Expresión infija : 2 * 4

Expresión postfija : 3

Pila de operadores: +

Tabla de prioridad

Simbolo	Off Stack Prioridad	On Stack Prioridad
+	1	1
-	1	1
*	2	2
/	2	2
^	4	3
(	5	0

Ejemplo Simple

Expresión infija : * 4
Expresión postfija : 3 2
Pila de operadores: +

Tabla de prioridad

Simbolo	Off Stack Prioridad	On Stack Prioridad
+	1	1
-	1	1
*	2	2
/	2	2
^	4	3
(	5	0

Ejemplo Simple

Expresión infija : 4

Expresión postfija : 3 2

Pila de operadores: + *

Tabla de prioridad

Simbolo	Off Stack Prioridad	On Stack Prioridad
+	1	1
-	1	1
*	2	2
/	2	2
^	4	3
(	5	0

Ejemplo Simple

Expresión infija :

Expresión postfija : 3 2 4

Pila de operadores: + *

Tabla de prioridad

Simbolo	Off Stack Prioridad	On Stack Prioridad
+	1	1
-	1	1
*	2	2
/	2	2
^	4	3
(	5	0

Ejemplo Simple

Expresión infija :

Expresión postfija : 3 2 4 *

Pila de operadores: +

Tabla de prioridad

Simbolo	Off Stack Prioridad	On Stack Prioridad
+	1	1
-	1	1
*	2	2
/	2	2
^	4	3
(	5	0

Ejemplo Simple

Expresión infija :

Expresión postfija : 3 2 4 * +

Pila de operadores:

Tabla de prioridad

Simbolo	Off Stack Prioridad	On Stack Prioridad
+	1	1
-	1	1
*	2	2
/	2	2
^	4	3
(	5	0

Otro Ejemplo

infija

3 * 4 / (3 + 1)

posfija

3 4 * 3 1 + /

pila

Operador	P.Expresión	P.Pila
^	4	3
* /	2	2
+ -	1	1
(	5	0
)	NA	NA

Los pasos a seguir para transformar una expresión algebraica de notación infija a postfija son:

1. Obtener caracteres de la expresión y repetir los pasos 2 al 4 para cada carácter.
2. Si es un operando, pasarlo a la expresión *postfija*.
3. Si es operador:
   - 3.1. Si la pila está vacía, meterlo en la pila. Repetir a partir de 1.
   - 3.2. Si la pila no está vacía:

     Si la prioridad del operador leído es mayor que la prioridad del operador cima de la pila, meterlo en la pila y repetir a partir de 1.

     Si la prioridad del operador es menor o igual que la prioridad del operador de la cima de la pila, sacar el operador cima de la pila y pasarlo a la expresión postfija, volver a 3.
4. Si es paréntesis derecho:
   - 4.1. Sacar el operador cima y pasarlo a la expresión postfija.
   - 4.2. Si el nuevo operador cima es paréntesis izquierdo, suprimir el elemento cima.
   - 4.3. Si la cima no es paréntesis izquierdo, volver a 4.1.
   - 4.4. Volver a partir de 1.
5. Si quedan elementos en la pila, pasarlos a la expresión postfija.
6. Fin del algoritmo.

Infija a Postfija

- Convertir las sgtes. expresiones infijas a expresiones postfija:

  2 ^ 3 ^ 3 + 5 * 1

  11 + 2 - 1 * 3 / 3 + 2 ^ 2 / 3

Simbolo	Off Stack Prioridad	On Stack Prioridad
+	1	1
-	1	1
*	2	2
/	2	2
^	4	3
(	5	0

Los pasos a seguir para transformar una expresión algebraica de notación infija a postfija son:

1. Obtener caracteres de la expresión y repetir los pasos 2 al 4 para cada carácter.
2. Si es un operando, pasarlo a la expresión *postfija*.
3. Si es operador:
   - 3.1. Si la pila está vacía, meterlo en la pila. Repetir a partir de 1.
   - 3.2. Si la pila no está vacía:

     Si la prioridad del operador leído es mayor que la prioridad del operador cima de la pila, meterlo en la pila y repetir a partir de 1.

     Si la prioridad del operador es menor o igual que la prioridad del operador de la cima de la pila, sacar el operador cima de la pila y pasarlo a la expresión postfija, volver a 3.
4. Si es paréntesis derecho:
   - 4.1. Sacar el operador cima y pasarlo a la expresión postfija.
   - 4.2. Si el nuevo operador cima es paréntesis izquierdo, suprimir el elemento cima.
   - 4.3. Si la cima no es paréntesis izquierdo, volver a 4.1.
   - 4.4. Volver a partir de 1.
5. Si quedan elementos en la pila, pasarlos a la expresión postfija.
6. Fin del algoritmo.

Ejercicio

1 - 2 ^ 3 ^ 3 - (4 + 5 * 6) * 7

Paso 2 - Ejercicio de pila/cola

- Una expresión *postfija* es una expresión matemática pero con los operadores escritos después de los operandos en lugar de antes.
  `1 + 1` se convierte en `1 1 +`
  `1 + 2 * 3 + 4` se convierte en `1 2 3 * + 4 +`

  – soportado por muchos tipos de calculadoras de lujo
  – nunca es necesario usar paréntesis
  – nunca necesita usar un carácter = para evaluar en una calculadora

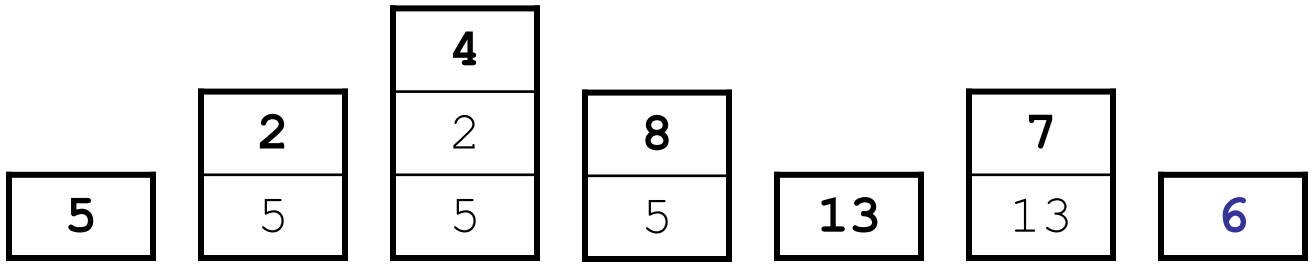
- Escriba un método `postfixEvaluate` que acepte una cadena de expresión postfija, la evalúe y devuelva el resultado.
  – Todos los operandos son enteros; los operadores legales son `+` , `-` , `*` y `/`
  `postFixEvaluate("5 2 4 * + 7 -")` devuelve `6`

Paso 2 - Algoritmo Postfija

- El algoritmo: usar una **pila**
  - Cuando vea un operando, empújelo a la pila.
  - Cuando vea un operador:
    - sacar los dos últimos operandos de la pila.
    - aplicarles el operador.
    - empujar el resultado a la pila.
  - Cuando haya terminado, el único elemento restante de la pila es el resultado.

"5 2 4 * + 7 -"

5 2 4 * + 7 -



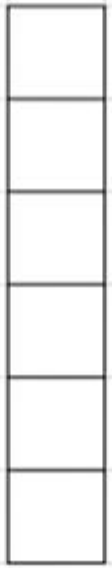
Ejemplo – evaluar postfija

infija

3	*	4	/	(	3	+	1	)

posfija

3	4	*	3	1	+	/



pila

Ope1		Ope2	
3	*	4	12
3	+	1	4
12	/	4	3

Operador	P.Expresión	P.Pila
^	4	3
* /	2	2
+ -	1	1
(	5	0
)	NA	NA

Los pasos a seguir para transformar una expresión algebraica de notación infija a postfija son:

1. Obtener caracteres de la expresión y repetir los pasos 2 al 4 para cada carácter.
2. Si es un operando, pasarlo a la expresión *postfija*.
3. Si es operador:
   - 3.1. Si la pila está vacía, meterlo en la pila. Repetir a partir de 1.
   - 3.2. Si la pila no está vacía:

     Si la prioridad del operador leído es mayor que la prioridad del operador cima de la pila, meterlo en la pila y repetir a partir de 1.

     Si la prioridad del operador es menor o igual que la prioridad del operador de la cima de la pila, sacar el operador cima de la pila y pasarlo a la expresión postfija, volver a 3.
4. Si es paréntesis derecho:
   - 4.1. Sacar el operador cima y pasarlo a la expresión postfija.
   - 4.2. Si el nuevo operador cima es paréntesis izquierdo, suprimir el elemento cima.
   - 4.3. Si la cima no es paréntesis izquierdo, volver a 4.1.
   - 4.4. Volver a partir de 1.
5. Si quedan elementos en la pila, pasarlos a la expresión postfija.
6. Fin del algoritmo.

Ejemplos

- Cuál es el resultado de evaluar la sgte. expresión postfija (polaca inversa)?

  6 3 2 + *

A. 18
B. 36
C. 24
D. 11
E. 30

- Evaluar las sgtes. expresiones postfija y escribir su correspondiente expresion infija:

  2 3 2 4 * + * 1 2 3 4 ^ * +

  1 2 - 3 2 ^ 3 * 6 / + 2 5 ^ 1 -

Solución del ejercicio

```
//Evalúa la expresión postfija dada y devuelve su resultado.
//Precondición: la cadena representa una expresión postfija
  válida
public static int postfixEvaluate(String expression) {
    Stack<Integer> s = new Stack<Integer>();
    Scanner input = new Scanner(expression);
    while (input.hasNext()) {
        if (input.hasNextInt()) {    // un operando (entero)
            s.push(input.nextInt());
        } else {                     // un operador
            String operator = input.next();
            int operand2 = s.pop();
            int operand1 = s.pop();
            if (operator.equals("+")) {
                s.push(operand1 + operand2);
            } else if (operator.equals("-")) {
                s.push(operand1 - operand2);
            } else if (operator.equals("*")) {
                s.push(operand1 * operand2);
            } else {
                s.push(operand1 / operand2);
            }
        }
    }
    return s.pop();
}
```

Ejercicio

infija

4	*	(	5	+	6	-	(	8	/	2	^	3	)	-	7	)	-	1

posfija

Ope1 Ope2

pila

O	PE	PP
^	4	3
* /	2	2
+ -	1	1
(	5	0
)	NA	NA

Estudio de caso: Evaluador de expresiones

Algoritmo de paso de notación infija a postfija

Los pasos a seguir para transformar una expresión algebraica de notación infija a postfija son:

1. Obtener caracteres de la expresión y repetir los pasos 2 al 4 para cada carácter.
2. Si es un operando, pasarlo a la expresión *postfija*.
3. Si es operador:
   - 3.1. Si la pila está vacía, meterlo en la pila. Repetir a partir de 1.
   - 3.2. Si la pila no está vacía:

     Si la prioridad del operador leído es mayor que la prioridad del operador cima de la pila, meterlo en la pila y repetir a partir de 1.

     Si la prioridad del operador es menor o igual que la prioridad del operador de la cima de la pila, sacar el operador cima de la pila y pasarlo a la expresión postfija, volver a 3.
4. Si es paréntesis derecho:
   - 4.1. Sacar el operador cima y pasarlo a la expresión postfija.
   - 4.2. Si el nuevo operador cima es paréntesis izquierdo, suprimir el elemento cima.
   - 4.3. Si la cima no es paréntesis izquierdo, volver a 4.1.
   - 4.4. Volver a partir de 1.
5. Si quedan elementos en la pila, pasarlos a la expresión postfija.
6. Fin del algoritmo.

Evaluación de expresiones Postfija

- Fácil de hacer con pilas
- Dada una expression postfija correcta:
  - Obtener el próximo caracter o token
  - Si es un operando, ponerlo(push) en la pila
  - Si es un operador:
    - Sacar (pop) el contenido de la pila, y considerarlo como el operando del lado derecho
    - Sacar (pop) el sgte. contenido de la pila, y considerarlo como el operando del lado izquierdo
    - Aplica el operador a los dos operandos
    - Poner(push) el resultado hacia la pila
  - cuando la expresión se ha agotado, el resultado es la parte superior (y único elemento) de la pila

Evaluación de expresiones Postfija

Al describir el algoritmo, `expsion` es la cadena con la expresión *postfija*. El número de elementos es la longitud, `n`, de la cadena. Los pasos a seguir son los siguientes:

1. Examinar `expsion` elemento a elemento: repetir los pasos 2 y 3 para cada elemento.
2. Si el elemento es un operando, meterlo en la pila.
3. Si el elemento es un operador, se simboliza con `&`, entonces:
   - Sacar los dos elementos superiores de la pila, que se denominarán `b` y `a` respectivamente.
   - Evaluar `a & b`, el resultado es `z = a & b`.
   - El resultado `z`, meterlo en la pila. Repetir a partir del paso 1.
4. El resultado de la evaluación de la expresión está en el elemento cima de la pila.
5. Fin del algoritmo.

Algoritmo en Java – Clase Pila

```java
package evaluadorexpresiones;
public class Pila {
    private int n;
    private int tope;
    private Object pila[];
    public Pila(int n)
    {
        this.n=n;
        this.tope=0;
        this.pila=new Object[n];
    }
    public boolean estaVacia(){
        return tope==0;
    }
    public boolean estaLlena(){
    return tope==n;
    }
    public boolean apilar(Object dato){
        if(estaLlena()){
            return false;
        }
        pila[tope]=dato;
        tope++;
        return true;
    }
    public Object desapilar(){
        if(estaVacia()){
            return null;
        }
        tope--;
        return pila[tope];
    }
    public Object elementoTope(){
        return pila[tope-1];
    }
}
```

Algoritmo en Java – Clase Evaluador

```java
package evaluadorexpresiones;
public class Evaluador {
    public static double evaluar(String infija) {
        String posfija = convertir(infija);
        System.out.println("La expresión posfija es: " + posfija);
        return evaluarPosfija(posfija);
    }
    private static String convertir(String infija) {
        String posfija = "";
        Pila pila = new Pila(100);
        for (int i = 0; i < infija.length(); i++) {
            char letra = infija.charAt(i);
            if (esOperador(letra)) {
                if (pila.estaVacia()) {
                    pila.apilar(letra);
                } else {
                    int pe = prioridadEnExpresion(letra);
                    int pp = prioridadEnPila((char) pila.elementoTope());
                    if (pe > pp) {
                        pila.apilar(letra);
                    } else {
                        if (letra == ')') {
                            while ((char) pila.elementoTope() != '(') {
                                posfija += pila.desapilar();
                            }
                            pila.desapilar();
                        } else {
                            posfija += pila.desapilar();
                            pila.apilar(letra);
                        }
                    }
                }
            } else {
                posfija += letra;
            }
        }
        while (!pila.estaVacia()) {
            posfija += pila.desapilar();
        }
        return posfija;
    }
```

Algoritmo en Java – Clase Evaluador

```
42      private static double evaluarPosfija(String posfija) {
43          Pila pila = new Pila(100);
44          for (int i = 0; i < posfija.length(); i++) {
45              char letra = posfija.charAt(i);
46              if (!esOperador(letra)) {
47                  double num = new Double(letra + "");
48                  pila.apilar(num);
49              } else {
50                  double num2 = (double) pila.desapilar();
51                  double num1 = (double) pila.desapilar();
52                  double num3 = operacion(letra, num1, num2);
53                  pila.apilar(num3);
54              }
55          }
56          return (double) pila.elementoTope();
57      }
```

Algoritmo en Java – Clase Evaluador

```
58      private static boolean esOperador(char letra) {
59          if (letra == '*' || letra == '/' || letra == '+'
60                  || letra == '-' || letra == '(' || letra == ')' || letra == '^') {
61              return true;
62          } else {
63              return false;
64          }
65      }
66      private static int prioridadEnExpresion(char operador) {
67          if (operador == '^') {
68              return 4;
69          }
70          if (operador == '*' || operador == '/') {
71              return 2;
72          }
73          if (operador == '+' || operador == '-') {
74              return 1;
75          }
76          if (operador == '(') {
77              return 5;
78          }
79          return 0;
80      }
```

Algoritmo en Java – Clase Evaluador

```java
81      private static int prioridadEnPila(char operador) {
82          if (operador == '^') {
83              return 3;
84          }
85          if (operador == '*' || operador == '/') {
86              return 2;
87          }
88          if (operador == '+' || operador == '-') {
89              return 1;
90          }
91          if (operador == '(') {
92              return 0;
93          }
94          return 0;
95      }
```

```java
96      private static double operacion(char letra, double num1, double num2) {
97          if (letra == '*') {
98              return num1 * num2;
99          }
100         if (letra == '/') {
101             return num1 / num2;
102         }
103         if (letra == '+') {
104             return num1 + num2;
105         }
106         if (letra == '-') {
107             return num1 - num2;
108         }
109         if (letra == '^') {
110             return Math.pow(num1, num2);
111         }
112         return 0;
113     }
114 }
```

Algoritmo en Java – Clase principal

```
1
2  package evaluadorexpresiones;
3
4  import java.util.Scanner;
5
6  public class EvaluadorExpresiones {
7
8
9      public static void main(String[] args) {
10         Scanner sc= new Scanner(System.in);
11         System.out.println("Digite expresión que desea evaluar:");
12         String infija=sc.next();
13         System.out.println("El resultado es: "+Evaluador.evaluar(infija));
14
15     }
16
17 }
```