ESTRUCTURA: BI COLA

DEFINICIÓN. Una **BI-COLA** es una estructura de datos lineal y estática, que permite insertar y eliminar datos por ambos extremos.

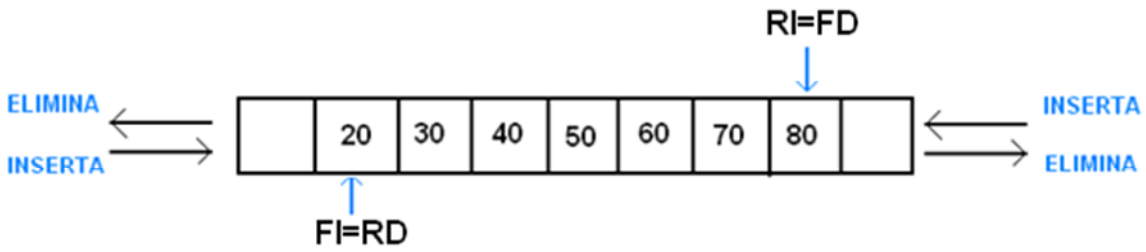
Es un mecanismo que integra en una única estructura las funcionalidades de las pilas y las colas.

CARACTERÍSTICAS DE UNA BI-COLA.

- Las eliminaciones, se los realiza por el frente, es decir por **FI(izquierda)** o **FD (derecha)**.
- Las inserciones, se los realiza por el final ya sea éste **RI(izquierda)** o **RD(derecha)**.
- Para las búsquedas y recorridos en las Bi-colas, se hace uso de una cola auxiliar.

REPRESENTACIÓN GRÁFICA DE UNA COLA CIRCULAR.

Se representara una bi-cola, por medio de vectores a las que se las conoce también como array's unidimensionales.



ELEMENTOS DE UNA BI-COLA.

MaxBiCola: Variable que contiene la capacidad de la bi-cola.

BICOLA[]: Vector que almacena los datos de la bi-cola

FI: puntero que apunta al primer dato por lado izquierdo de la bi-cola.

RI: puntero que apunta al último dato por lado izquierdo de la bi-cola.

FD: puntero que apunta al primer dato por lado derecho de la bi-cola.

RD: puntero que apunta al último dato por lado derecho de la bi-cola.

IMPLEMENTACIÓN DE LA ESTRUCTURA BI-COLA EN JAVA.

Con fines didácticos la implementación se realiza a través de dos clases: ***Principal y BiCola.***

I. CLASE PRINCIPAL EN JAVA.

Crea un objeto BC1 (Bi-Cola) y se ejecuta las operaciones que se han definido sobre esa estructura de datos.

```
public class Principal {

     public static void main(String[] args) {

     BiCola BC1=new BiCola ();

     // crea el objeto bi-cola.

     // Agrega código necesario para resolución de
     // problemas con colas

     }
}
```

II. CLASE BI-COLA EN JAVA.

Define los atributos y los métodos propios de una Bi-Cola.

```
public class BiCola {
     // Agregue aquí los atributos y métodos que
     // conformará la estructura de datos BI-COLA
}
```

ATRIBUTOS DE LA ESTRUCTURA COLA CIRCULAR.

Se define los elementos de la cola, como son las variables con los que se implementará esta nueva estructura:

```
public class BiCola {

 final int MaxBiCola = 100;  // define el tamaño de la cola
 int[] BICOLA;            // define un vector  que almacenara
                                los datos
 int FI, FD;         // define los punteros frente

 int RI, RD;         // y final de la cola
//Agregue aquí los métodos (operaciones) que se desarrollen
 // en clases.

}
```

1. OPERACIONES ELEMENTALES O DE ESTADO.

a. *CREAR E INICIALIZAR BI-COLA [CONSTRUCTOR].*

```
    public BiCola() {
     BCOLA = new int[MaxBiCola];
     FI = -1;  FD = -1;
     RI = -1;  RD = -1;
}
```

b. ***COLA VACÍA***, Devuelve verdadero si no existe ningún elemento en la bi-cola.

```
    public boolean colaVacia() {
    return ((FI == -1) && (RI == -1) && (FD == -1)
            && (RD == -1));
}
```

c. ***TAMAÑO DE LA BI-COLA***. Devuelve el número de elementos que posee la bi-cola

```
public int TamañoCola() {
     return (RI - FI) + 1;
}
```

d. ***COLA LLENA EXTREMO DERECHO***. Devuelve verdadero si la bi-cola está llena, cuando se lee de IZQUIERDA a DERECHA

```
public boolean ColaLlenaRI() {
     return RI == (MaxBiCola – 1);
}
```

e. ***COLA LLENA EXTREMO IZQUIERDA***. Devuelve verdadero si la bi-cola está llena, cuando se lee de DERECHA a IZQUIERDA

```
public boolean ColaLlenaRD() {
     return (RD - 1) < 0;
}
```

OPERACIONES FUNDAMENTALES

a. ***INSERTAR ELEMENTO POR EXTREMO DERECHO***. Inserta un elemento por el FINAL IZQUIERDO en la bi-cola; siempre y cuando se pueda, caso contrario despliega un mensaje de error.

```
public void InsertarRI(int x) {
     if (ColaLlenaRI()) {
               System.out.println("Cola Llena! No se pudo
               adicionar por FINAL IZQUIERDA.");
     } else {
          if (colaVacia()) {
               RI = RI + 1;
               BICOLA[RI] = x;
               FI = 0; FD = 0; RD = 0;
          } else {
               RI = RI + 1;
               BICOLA[RI] = x;
               FD = RI;
          }
     }
}
```

b. ***INSERTAR ELEMENTO POR EXTREMO IZQUIERDO.*** Inserta un elemento por el FINAL DERECHO en la bi-cola. Si la bi-cola esta vacia y si existe elemento en posición 0, muestra un mensaje de error.

```
public void InsertarRD(int x) {
     if (colaLlenaDI()) {
               System.out.println("Cola Llena! No se pudo
               adicionar.");
     } else {
          if (colaVacia()) {
               System.out.println("No se pudo adicionar
               por elementos FINAL DERECHO.");
          } else {
               RD = RD - 1;
               BICOLA[RD] = x;
               FI = RD; }
          }
}
```

c. ***ELIMINAR ELEMENTO POR EXTREMO IZQUIERDO***. Elimina un elemento por FRENTE IZQUIERDO de la bi-cola.

```
public int eliminarFI() {
     int x = -1;
     if (colaVacia()) {
               System.out.println("Cola Vacia! No se pudo
               eliminar.");
     } else {
          if (RI == FI) {
               x = BICOLA[FI];
               FI = -1; RI = -1;
               FD = -1; RD = -1;
          } else {
               x = BICOLA[FI];
               FI = FI + 1;
```

```
                RD = FI;
            }
        }
        return x;
}
```

d. ***ELIMINAR ELEMENTO POR EXTREMO DERECHO***. Elimina un elemento de la bi – cola por el frente derecho.

```
public int eliminarFD() {
     int x = -1;
     if (colaVacia()) {
              System.out.println("Cola Vacía! No se pudo
              eliminar.");
     } else {
          if (RD == FD) {
               x = BICOLA[FD];
               FI = -1;  RI = -1;
               FD = -1;  RD = -1;
          } else {
               x = BICOLA[FD];
               FD = FD - 1;
               RI = FD;
          }
     }
     return x;
}
```

e. ***RECORRER BI-COLA DE IZQUIERDA A DERECHA.*** Muestra todos los elementos de la bi-cola de IZQUIERDA A DERECHA,

siempre y cuando existiera elemento alguno caso contrario muestra un mensaje de error.

```
public void mostrarID() {
     if (colaVacia()) {
               System.out.println("Cola Vacía! No se
               puede mostrar nada.");
     } else {
          bcola aux = new bcola();
          System.out.print("COLA: ");
          while (!colaVacia()) {
               int x = eliminarFI();
               System.out.print(x + " ");
               aux.insertarRI(x);
          }
          while (!aux.colaVacia()) {
               insertarRI(aux.eliminarFI());
          }
     }
}
```

f. ***RECORRER BI-COLA DE DERECHA A IZQUIERDA.*** Muestra todos elementos de la bi-cola de DERECHA A IZQUIERDA, si existen caso contrario despliega un mensaje de error.

```
public void mostrarDI() {
     if (colaVacia()) {
               System.out.println("Cola Vacía! No se
               puede mostrar nada.");
     } else {
          bcola aux = new bcola();
          System.out.print("BICOLA: ");
```

```
            while (!colaVacia()) {
                int x = eliminarFD();
                System.out.print(x + " ");
                aux.insertarRI(x);
            }
            while (!aux.colaVacia()) {
                insertarRI(aux.eliminarFD());
            }
        }
```

g. ***BÚSQUEDA DE ELEMENTOS EN UNA BI COLA.*** Muestra la ubicación (índice) de un elemento dentro de la bi-cola, así como el total de las ocurrencias, si se los encuentra; caso contrario despliega un mensaje de Error.

```
public void Buscar() {
int pos=-1;
int c=0;
Scanner in = new Scanner(System.in);
System.out.println("Ingrese el valor a Buscar:”);
int eb= in.nextInt();
    if (ColaVacia()) {
            System.out.println("Bi-Cola Vacia! No se
            puede realizar Búsqueda de ningún
            elemento.");
    } else {
        BiCola BCaux = new BiCola();
        while (!ColaVacia()) {
        pos = FC;
        int x = EliminarFI();
            if (x == eb) { c=c+1;
            System.out.println("posición: " + pos);
```

```
                BCaux.InsertarRI(x);}
                else{ BCaux.InsertarRI(x);}
            }
            while (!BCaux.colaVacia()) {
                InsertarRI(BCaux.EliminarFI());
            }
        }
        if (c == 0)) { System.out.println("No…!, no existe
                el elemento en la Cola Circular.");

    } else { System.out.println(" Se encontró: "+ c +
            "elementos en la cola circular");
            }
}
```

2. OPERACIONES COMPLEMENTARIAS

a. ***INSERTAR N ELEMENTOS EN LA BI-COLA***. Permite insertan N elementos por el FINAL IZQUIERDO en la bi-cola.

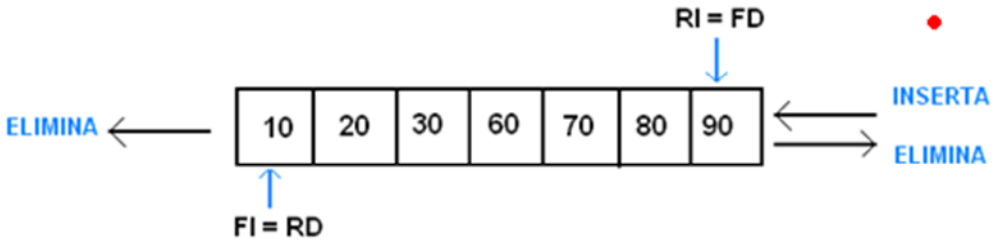
```
public void insertarNElementos() {
    Scanner in = new Scanner(System.in);
    System.out.print("Nro. Elementos: ");
    int n = in.nextInt();
    System.out.println("Ingrese elementos:");
    for (int i = 0; i < n; i++) {
        int x = in.nextInt();
        insertarRI(x);
    }
}
```

VARIANTES DE LAS BI-COLAS

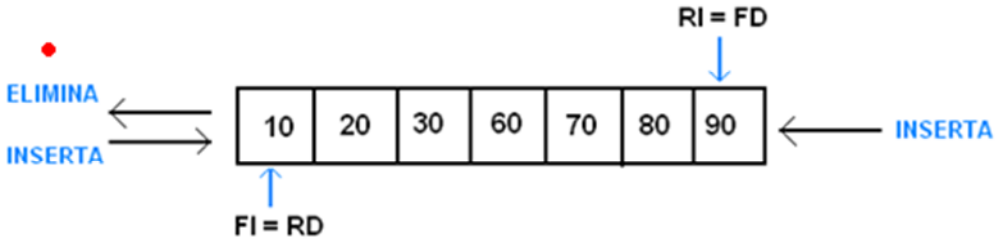
Es posible definir restricciones en una bicola con respecto al tipo de entrada o el tipo de salida de datos.

Existen dos variantes de la doble cola:

a. ***DOBLE COLA DE ENTRADA RESTRINGIDA***. Una bicola con entrada restringida es aquella que solo permite inserciones por uno de los extremos [FINAL]. Pero realiza las eliminaciones por los dos extremos.



b. ***DOBLE COLA DE SALIDA RESTRINGIDA***. Una bicola con restricción en la salida es aquella que admite inserciones por los dos extremos. Pero solo elimina datos por uno de los extremos [FRENTE].



TAREA NRO.5

a. Implemente la clase BiColaEntradaRestrungida.

b. Implemente la clase BiColaSalidaRestribgida.