ESTRUCTURA DE DATOS COLAS

Ing. Jaider de la Rosa Bertel

Docente_ingenieria3@uajs.edu.co

DESARROLLO

CONCEPTO DE COLAS:

Las colas son una estructura de datos que almacenan un conjunto de valores para los cuales las operaciones de agregar y eliminar datos se realizan en por los extremos opuestos de la estructura, de modo que los primeros elementos en ser ingresados en una cola son los primeros en ser extraídos o eliminados de ella; y así mismo los últimos elementos en ser agregados serán los últimos en ser sacados o eliminados.

A las colas se les conoce también como listas FIFO (First In, First Out) primero en entrar, primero en salir (Zahonero y Joyanes Aguilar, 1999). Un ejemplo puede ser, el caso de la cola de atención de un banco o la cola para comprar las boletas para entrar a un juego de fútbol, en donde la primera persona en llegar es la primera en ser atendida y en consecuencia la primera en salir de la cola.

La estructura de datos cola, no tiene un medio propio con el cual almacenar sus datos, por lo que depende de otra estructura auxiliar para hacerlo. El proceso empleado para representar o guardar los datos de una cola se le llama implantación de la cola. Para esto lo más habitual es usar arreglos o listas enlazadas que permitan guardar los datos de la cola; de manera que la cola procesa los datos del arreglo o la lista de una forma particular; haciendo que las operaciones de agregar y eliminar datos, se hagan en extremos opuestos de la estructura con la cual se implanta.

Una cola se caracteriza por tener un **frente** y un **final** que definen los extremos de la misma, siendo el frente el punto por donde se realizan las eliminaciones y el final el extremo por el cual se agregan nuevos elementos dentro de la cola. Una tercera característica de la cola es que posee un tamaño, que determina la capacidad de almacenamiento o número de elementos máximos contendidos dentro de la cola.

Según la cantidad de elementos actualmente contenidos en una cola, esta puede estar vacía si no contiene datos; llena cuando el número de elementos es igual al tamaño máximo definido y por supuesto el estado intermedio, ni vacía ni llena, que de hecho es de poco interés. En cuanto a las operaciones que comúnmente se hacen con una cola, están:

- Agregar nuevos elementos a la cola (por el final).
- Eliminar elementos de la cola (por el frente).
- Consultar el elemento del frente de la cola.
- Consultar el elemento del final de la cola.

La posibilidad de efectuar alguna de estas operaciones depende del estado de la cola, de manera que para poder eliminar o consultar el elemento del frente de la cola es necesario que la cola no esté vacía; lo cual también se requiere para consultar el elemento del final de la cola. Igualmente

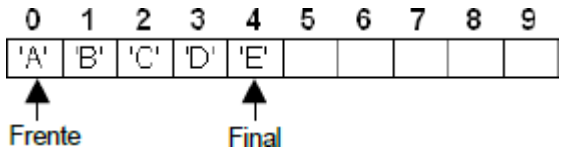
para agregar un nuevo elemento a la cola se requiere que no esté llena. El acceso a los elementos de la cola es restringido, puesto que solo se permite acceder exclusivamente al elemento del frente y al elemento del final de la cola; fuera de la estructura no hay acceso a los elementos "intermedios" de la cola; al menos que se procesen los datos y se muevan a otra cola auxiliar.

La implementación de las operaciones sobre la cola puede variar dependiendo de la estructura seleccionada para implementarla (arreglos o listas), pues el modo en que se representan las características de frente, final y tamaño es distinto para arreglos y para listas, así como la codificación de las operaciones de agregar y eliminar.

IMPLEMENTACIÓN CON ARREGLOS:

En la implementación utilizando arreglos, se tiene que frente y final pueden representarse con números enteros, que indican la posición en el arreglo de los elementos que están en los extremos de la estructura. La naturaleza de los datos almacenados en la cola queda determinada por el tipo de datos definido para el arreglo que se utiliza como estructura auxiliar en la implementación; y el tamaño o capacidad de la cola estará asociado al tamaño del arreglo.

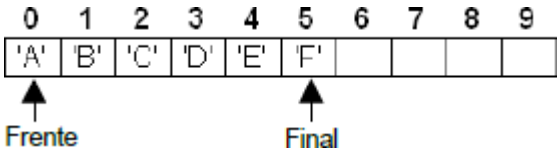
Para la implementación con arreglos el frente de la cola siempre será la posición cero del arreglo y que final varía según las inserciones y eliminaciones que se hagan sobre la estructura.



Cuando se agrega un nuevo dato a la cola; por ejemplo agregar la letra F, se deben ejecutar los siguientes pasos:

- Verificar que la cola no esté llena (Final<Tamaño).
- Aumentar final en uno, es decir: final = final + 1.
- Guardar en la posición de final el dato, es decir: Arreglo[final] = 'F'.

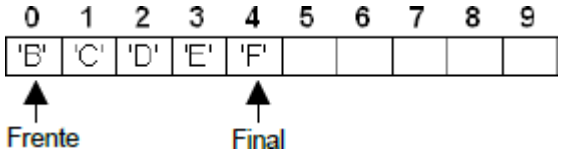
Con lo cual la estructura queda de la siguiente manera:



Cuando se elimina, siempre se quita el elemento del frente. En este caso siempre será el de la posición cero (implementación con vectores), entonces lo que se debe hacer es desplazar los elementos del arreglo un puesto hacia la izquierda, empezando desde el elemento de la posición 1 y continuando hasta el extremo derecho (final); y después se disminuye en uno el valor de final. Los pasos para esta operación son los siguientes:

- Verificar que la cola no esté vacía (Final>=0).
- Desplazar hacia la izquierda los elementos del arreglo desde la posición uno hasta la posición de final, es decir, hacer Arreglo[i-1] = Arreglo[i]; con i variando desde 1 hasta final.
- Disminuir final en uno, es decir, hacer final = final – 1.

Siendo así las cosas, la estructura queda de la siguiente manera:



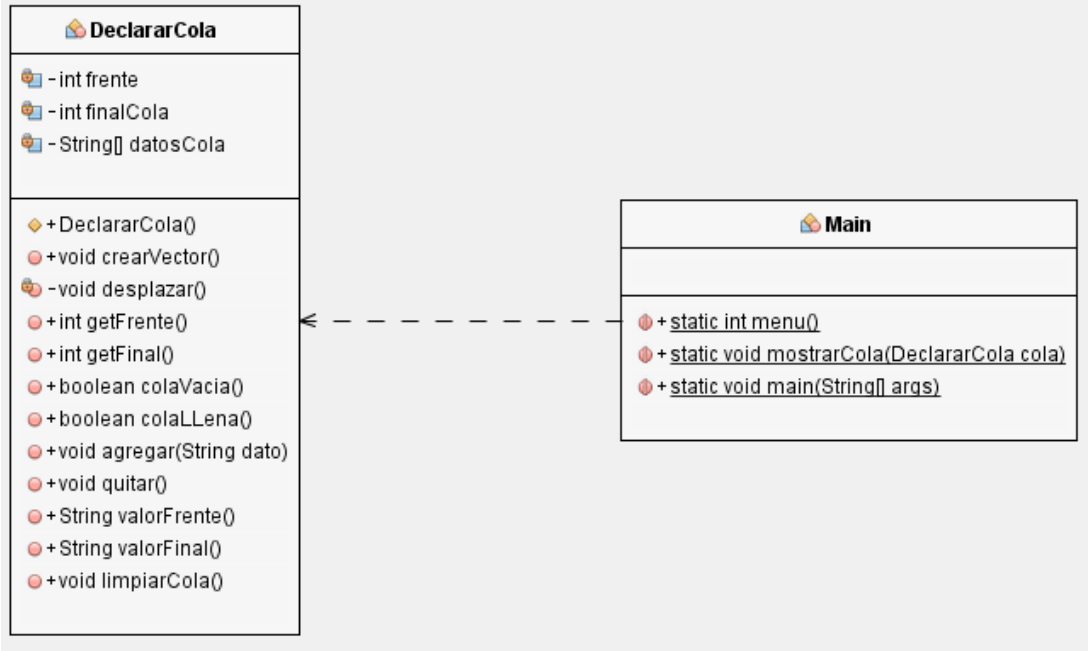
EJERCICIO PROPUESTO CON ARREGLOS:

A continuación, se implementa un ejercicio que permite almacenar en una cola datos de tipo cadena, el ejercicio en su implementación permite al usuario las siguientes opciones:

- Agregar datos a la cola.
- Eliminar datos de la cola.
- Consultar la información del frente y final de la cola.
- Procesar la información almacenada en la cola.

Para la solución del ejercicio, se implementara la cola utilizando como estructura auxiliar un vector, posteriormente se realizara la implementación con listas. Se utilizaran dos ficheros, en donde se implementan las operaciones de agregar y quitar elementos de la cola; así como procesar la información de la cola.

- **Diseño de clases UML de la solución**



- **Implementación de la clase DeclararCola en el fichero DeclararCola.java**

En el fichero **DeclararCola.java** se declara la clase DeclararCola, que tendrá los métodos que permiten realizar las operaciones de agregar, quitar y consultar el valor del frente y final de la cola.

```
public class DeclararCola {
    static final int N = 10; //Número máximo de elementos que se podrán almacenar en la cola.
    private int frente; //Atributo para hacer referencia a la posición del primer elemento de la cola.
    private int finalCola; //Atributo para hacer referencia a la posición del último elemento de la cola.
//Vector que se usara como estructura auxiliar para guardar los elementos de la cola.
    private String[] datosCola = new String[N];
//Se declara el método constructor de la clase.
    public DeclararCola(){
        frente = 0 ; //Asignamos al frente la primera posición del arreglo.
        finalCola = -1; //La cola está vacía al principio, cuando la posición del final es -1.
    }
    public void crearVector(){ //No se utiliza.
        datosCola = new String[N];
    }
//Se declara un método auxiliar para mover los elementos del vector de derecha a izquierda.
//El método será de uso interno en la clase (privado), este se llama cuando se quita el elemento del
//frente y se quieren correr los demás elementos de la cola hacia el frente de la misma.
    private void desplazar(){
        for (int i=1; i<= getFinal(); i++){
//Cada elemento de la derecha (i) pasa a su izquierda inmediata(i-1).
            datosCola[i-1] = datosCola[i];
        }
    }
```

```
//Declaración del método que retorna la posición del frente de la cola.
   public int getFrente(){
      return frente; //Retornamos la posición del frente.
   }
//Declaración del método que retorna la posición del final de la cola.
   public int getFinal(){
      return finalCola; //Retornamos la posición del final.
   }
//Métodos para determinar los estados de la cola (vacía o llena), si la cola está llena retorna un
//valor de true; de lo contrario retorna el valor false. De igual forma si la pila está vacía retorna el
//valor true.
   public boolean colaVacia(){
//Se debe tener en cuenta que final indica cuantos elementos actualmente tiene la cola. La cola
//está vacía si final es igual a -1.
      if (getFinal() == -1){
         return true;
      }else{
         return false;
      }
   }
//La cola estará llena si el final es igual al número máximo de elementos que pueda soportar la
//estructura de datos que se está utilizando, en este caso el número máximo de elementos del
//vector.
   public boolean colaLLena(){
      if (getFinal() >= N-1){
         return true;
      }else{
         return false;
      }
   }
//Declaración de los métodos para realizar las operaciones básicas sobre la cola (agregar y quitar).
//Implementación del método que permite agregar elementos al final de la cola, dato será el
//parámetro que se agregar a la cola.
   public void agregar(String dato){
      if (! colaLLena()){ //Si la cola no está llena
         finalCola = getFinal() + 1;
//Final apunta a una nueva posición en el vector para agregar el nuevo elemento.
         datosCola[getFinal()] = dato; //Se agrega el elemento por el final de la cola.
      }
   }
//Implementación del método que permite eliminar un elemento de la cola, desplazando hacia la izquierda
//todos sus elementos y disminuyendo el final de la cola en uno.
   public void quitar(){
      if (! colaVacia()){ //Si la cola no está vacía
         desplazar(); //Se corren una posición a la izquierda, todos los elementos de la cola.
         finalCola = getFinal() - 1;
//Se disminuye el valor de final en uno (se apunta a una posición anterior del vector).
      }
   }
//Declaración de los métodos para obtener los valores de los extremos de la cola (valor del frente y
//valor del final).
//Implementación del método que retorna el valor del elemento del frente de la cola.
   public String valorFrente(){
      return datosCola[getFrente()];
   }
   public String valorFinal(){
```

```
        return datosCola[getFinal()];
    }
//Método para eliminar todos los nodos de la cola.
    public void limpiarCola(){
        while (! colaVacia()){ //Mientras que la cola no este vacía.
            quitar(); //Se elimina el elemento del frente.
        }
    }
}
```

- **Implementación de la clase Main en el fichero Main.java**

En el fichero **Main.java** se declara la clase Main, que tendrá el método que permite visualizar los elementos de la cola y el menú de opciones para realizar las operaciones de agregar y quitar datos de la cola, así como mostrar los datos de la cola pasándolos a una cola auxiliar temporalmente para que no se pierdan los datos mientras se procesan.

```
public class Main {
//Método para asignar un menú de opciones de la aplicación.
    public static int menu(){
        int opcion = 0;
        do{
            opcion = Integer.parseInt(JOptionPane.showInputDialog("===== IMPLEMENTACIÓN DE COLAS
- OPCIÓN DEL MENÚ ===== \n\n"
            +"1. Agregar Datos a la Cola \n"+"2. Eliminar Datos de la Cola \n"+"3. Mostrar Datos de la Cola
\n"+
                    "4. Vaciar la Cola \n"+"5. Salir"+"\n \n Seleccione una opción del 1 al 5"));
        }while(opcion <= 0 || opcion > 5);
        return opcion;
    }

//Método para mostrar los datos de la cola.
    public static void mostrarCola(DeclararCola cola){
//Así se mueven los elementos de la cola, se quita el elemento del frente y se agrega por el final de
//otra cola auxiliar.
        DeclararCola temp = new DeclararCola();
        String verDatosCola = "";
        while (! cola.colaVacia()){ //Mientras la cola no este vacía se muestran sus datos.
            verDatosCola = verDatosCola+String.valueOf(" ----"+cola.valorFrente()+"\n");
            temp.agregar(cola.valorFrente()); //Se agrega el elemento del frente a una cola temporal.
            cola.quitar(); //Luego se quita el elemento del frente de la cola actual.
        }
        JOptionPane.showMessageDialog(null, "========= ELEMENTOS DE LA DE COLA
==========="+"\n"+verDatosCola+"\n");
       while (! temp.colaVacia()){ //Se pasan los datos de la cola temporal a la cola original.
            cola.agregar(temp.valorFrente());
            temp.quitar();
        }
    }

    public static void main(String[] args) {
        DeclararCola cola = new DeclararCola();
        String dato;
        int opcion;
```

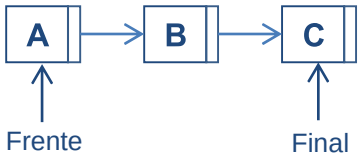
```
      do{
         opcion = menu();
         switch(opcion) {
         case 1:
            dato = JOptionPane.showInputDialog(null,"Digite el dato que quiere agregar a la Cola:");
            if (! cola.colaLLena()){ //Se agrega siempre que la cola no esté llena.
               cola.agregar(dato); //Se agrega a la cola el dato por el final.
               JOptionPane.showMessageDialog(null, "============ IMPLEMENTACIÓN DE COLAS
============"+"\n\n"+
                  "--- Dato agregado: "+cola.valorFinal());
               mostrarCola(cola);
            }else{
               JOptionPane.showMessageDialog(null, "============ IMPLEMENTACIÓN DE COLAS
============"+"\n\n"+
                     "La cola está llena, No puede agregar más elementos \n\n");
            }
            break;
         case 2:
            if (! cola.colaVacia()){ //Para poder eliminar el dato del frente, la cola no debe estar vacía.
               String elemento = cola.valorFrente();
               cola.quitar();
               JOptionPane.showMessageDialog(null, "============ IMPLEMENTACIÓN DE COLAS
============"+"\n\n"+
                     "--- Se eliminó el elemento: "+elemento+" de la cola");
               mostrarCola(cola);
            }else{
               JOptionPane.showMessageDialog(null, "============ IMPLEMENTACIÓN DE COLAS
============"+"\n\n"+
                     "La cola está vacía, No pueden quitar datos \n\n");
            }
            break;
         case 3:
            mostrarCola(cola);
            break;
         case 4:
            cola.limpiarCola();
            JOptionPane.showMessageDialog(null, "============ IMPLEMENTACIÓN DE COLAS
============"+"\n\n"+
                     "La cola está vacía, No pueden quitar datos \n\n");
            mostrarCola(cola);
            break;
         case 5:
            break;
         }
      }while(opcion != 5);
   }
}
```

IMPLEMENTACIÓN CON LISTAS ENLAZADAS:

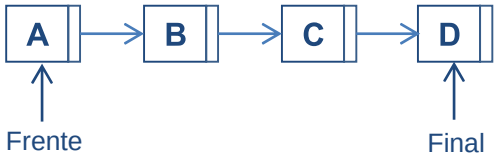
Como se mencionó, las pilas no contaban con una estructura de datos propia para almacenar la información; en este sentido su implementación se debe realizar utilizando una estructura de datos auxiliar, que puede ser un vector o una lista enlazada. En el ejercicio anterior se realizó la implementación de la cola utilizando un vector como estructura auxiliar, en esta parte se realizara la implementación usando listas enlazadas.

Para la implementación con listas, el frente y el final de la cola serán el primer y el último nodo de la lista, esto teniendo en cuenta que el primer nodo de una lista es la referencia que conocemos para agregar información (Allen Weiss, 2004). Independientemente de la forma como se implemente la cola el concepto será el mismo; entonces tendremos que agregar la información por el nodo del final y quitarla siempre por el nodo del frente. También se deben tener en cuenta los estados de la cola (llena o vacía) para realizar las operaciones de agregar o quitar elementos.

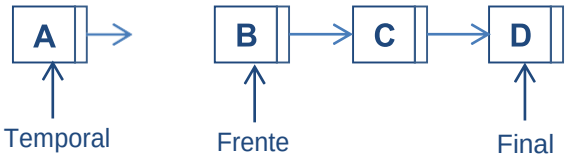
Cuando se agregue el primer nodo de la cola este será el nodo del frente y del final de la cola, para agregar un nuevo nodo se debe verificar el la cola no esté llena, en este caso se procede a agregar el nuevo nodo por el final de la misma, es decir después del primer nodo.

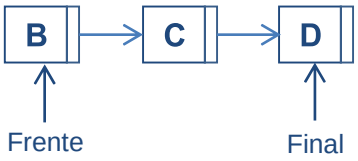


Si se quiere agregar un nuevo elemento a la cola, se verifica que la cola no esté llena, luego se crea un nuevo nodo con su respectiva información y se le dice al apuntador del último nodo, que apunte a donde está el nuevo nodo; finalmente se asigna el final al nuevo nodo que se agregó.



Para eliminar elementos de la cola se debe verificar que no esté vacía, entonces se procede a crear un apuntador temporal para que tome la referencia del frente; luego se asigna que el frente apunte al siguiente nodo del que esta de frente. Si seguimos la gráfica anterior frente es el nodo A, entonces ahora será el nodo B. para eliminar la referencia al nodo A, la cual llamamos temporal, simplemente se asigna nulo.





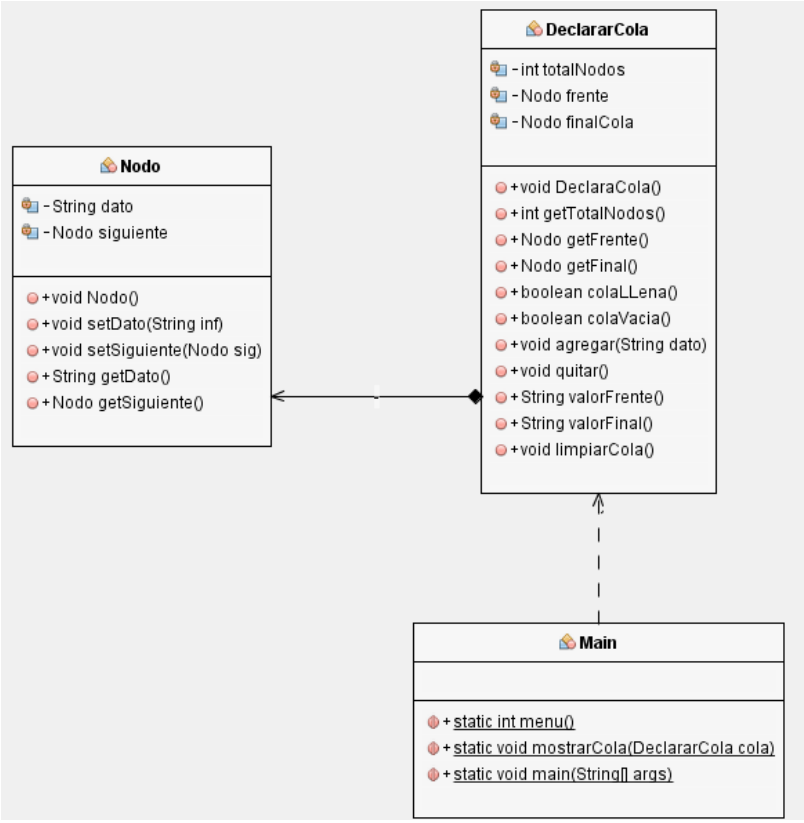
EJERCICIO PROPUESTO CON LISTAS ENLAZADAS:

A continuación se implementa un ejercicio que permite almacenar en una cola datos de tipo cadena, el ejercicio en su implementación permite al usuario las siguientes opciones:

- Agregar datos a la cola.
- Eliminar datos de la cola.
- Consultar la información del frente y final de la cola.
- Procesar la información almacenada en la cola.

Para la solución del ejercicio, se implementara la cola utilizando como estructura auxiliar una lista enlazada. Se utilizaran tres ficheros, en donde se implementan: la clase nodo, las operaciones de agregar y quitar elementos de la cola; así como procesar la información de la cola.

- **Diseño de clases UML de la solución**



- **Implementación de la clase Nodo en el fichero Nodo.java**

En el fichero **Nodo.java** se declara la clase Nodo, que tendrá como atributos la información y el apuntador del nodo que se agregara a la cola.

```
public class Nodo {
    private String dato;
    private Nodo siguiente;
    public void Nodo(){
        dato = "";
        siguiente = null;
    }
    public void setDato(String inf){
        dato = inf;
    }
    public void setSiguiente(Nodo sig){
        siguiente = sig;
    }
    public String getDato(){
        return dato;
    }
    public Nodo getSiguiente(){
        return siguiente;
    }
}
```

- **Implementación de la clase DeclararCola en el fichero DeclararCola.java**

En el fichero **DeclararCola.java** se declara la clase DeclararCola, que tendrá los métodos que permiten realizar las operaciones de agregar, quitar y consultar el valor del frente y final de la cola; así como eliminar toda la información.

```
public class DeclararCola {
 static final int cantidad_nodos = 5; //Número máximo de nodos que se podrán almacenar en la cola.
    private int totalNodos; //Atributo para contar los nodos agregados a la cola.
    private Nodo frente; //Apuntador que devuelve el nodo del frente de la cola.
    private Nodo finalCola; //Apuntador que devuelve el nodo del final de la cola.
//Método constructor de la clase.
    public void DeclaraCola(){
        totalNodos = 0;
        frente = null;
        finalCola = null;
    }
//Método que retorna el número de nodos agregados a la cola.
    public int getTotalNodos(){
        return totalNodos;
    }


//Método que retorna el apuntador al nodo del frente de la cola.
    public Nodo getFrente(){
        return frente;
```

```
    }
//Método que retorna el apuntador al nodo del final de la cola.
    public Nodo getFinal(){
        return finalCola;
    }
//Métodos para determinar si la cola está llena o vacía.
    public boolean colaLLena(){
        if(getTotalNodos() >= cantidad_nodos){
            return true;
        }else{
            return false;
        }
    }
    public boolean colaVacia(){
        if(getTotalNodos() == 0){
            return true;
        }else{
            return false;
        }
    }
//Método para agregar un dato en el campo de información, de un nuevo nodo.
    public void agregar(String dato){
        if (! colaLLena()){ //Si la cola no está llena, se pueden agregar nodos.
            Nodo temp = new Nodo(); //Se crea un nuevo objeto o instancia de la clase Nodo (temp).
            temp.setDato(dato); //Se asigna la información al respectivo atributo.
            if (getFrente() == null){ //Esta condición es para cuando se agrega el primer nodo de la cola.
                frente = temp;
            }else{
                getFinal().setSiguiente(temp); //Si hay nodos en la cola, el ultimo nodo apuntara al nuevo.
            }
            finalCola = temp;
            totalNodos = totalNodos + 1; //El contador de nodos se incrementa en uno.
        }
    }
//Elimina el nodo del frente de la cola.
    public void quitar(){
        Nodo temp;
        if (!colaVacia()){ //Si la cola no está vacía, se pueden eliminar nodos.
            temp = getFrente(); //Se asigna una referencia temporal al nodo del frente.
            frente = getFrente().getSiguiente(); //ahora frente, será el siguiente del nodo frente.
            if (getFrente() == null){ //Esta condición es para cuando se elimina el único nodo que hay.
                finalCola = null;
            }
            temp = null; //La referencia temporal se hace nulo, que era el anterior nodo del frente.
            totalNodos = totalNodos - 1; //El contador de nodos se decremento en uno.
        }
    }
//Método que muestra la información del nodo del frente de la cola.
    public String valorFrente(){
        return getFrente().getDato();
    }
//Método que muestra la información del nodo del final de la cola.
    public String valorFinal(){
        return getFinal().getDato();
    }
//Método para eliminar todos los nodos de la cola.
```

```
    public void limpiarCola(){
        while(! colaVacia()){
            quitar();
        }
    }
}
```

- **Implementación de la clase Main en el fichero Main.java**

En el fichero **Main.java** se declara la clase Main, que tendrá el método que permite visualizar los elementos de la cola y el menú de opciones para realizar las operaciones de agregar y quitar datos de la cola; así como mostrar los datos de la cola pasándolos a una cola auxiliar temporalmente para que no se pierdan los datos mientras se procesan.

```
public class Main {
    //Método para asignar un menú de opciones de la aplicación.
    public static int menu(){
        int opcion = 0;
        do{
            opcion = Integer.parseInt(JOptionPane.showInputDialog("===== IMPLEMENTACIÓN DE COLAS -
OPCIÓN DEL MENÚ ===== \n\n"
            +"1. Agregar Datos a la Cola \n"+"2. Eliminar Datos de la Cola \n"+"3. Mostrar Datos de la Cola \n"+
                "4. Vaciar la Cola \n"+"5. Salir"+"\n \n Seleccione una opción del 1 al 5"));
        }while(opcion <= 0 || opcion > 5);
        return opcion;
    }
//Método para mostrar los datos de la cola.
    public static void mostrarCola(DeclararCola cola){
//Así se mueven los elementos de la cola, se quita el elemento del frente y se agrega por el final.
        DeclararCola temp = new DeclararCola(); //Se crea un nuevo objeto de la clase cola.
        String verDatosCola = ""; //variable tipo cadena vacía para almacenar los datos.
        while (! cola.colaVacia()){ //Mientras la cola no este vacía se muestran sus datos.
            verDatosCola = verDatosCola+String.valueOf(" ----"+cola.valorFrente()+"\n");
            temp.agregar(cola.valorFrente()); //Se agrega el elemento del frente a una cola temporal.
            cola.quitar(); //Luego se quita el elemento del frente de la cola actual.
        }
        JOptionPane.showMessageDialog(null, "========= ELEMENTOS DE LA DE COLA
==========="+"\n"+verDatosCola+"\n");
       while (! temp.colaVacia()){ //Se pasan los datos a la cola temporal a la original.
            cola.agregar(temp.valorFrente());
            temp.quitar();
        }
    }

    public static void main(String[] args) {
        DeclararCola cola = new DeclararCola();
        String dato;
        int opcion;
        do{
            opcion = menu();
            switch(opcion) {
            case 1:
                dato = JOptionPane.showInputDialog(null,"Digite el dato que quiere agregar a la Cola:");
```

```
            if (! cola.colaLLena()){ //Se agrega siempre que la cola no esté llena.
                cola.agregar(dato); //Se agrega a la cola el dato por el final.
                JOptionPane.showMessageDialog(null, "============ IMPLEMENTACIÓN DE COLAS
============"+"\n\n"+
                    "--- Dato agregado: "+cola.valorFinal());
                mostrarCola(cola);
            }else{
                JOptionPane.showMessageDialog(null, "============ IMPLEMENTACIÓN DE COLAS
============"+"\n\n"+
                    "La cola está llena, No puede agregar más elementos \n\n");
            }
            break;
        case 2:
//Para poder eliminar el dato del frente, la cola esta no debe estar vacía.
            if (! cola.colaVacia()){
                String elemento = cola.valorFrente();
                cola.quitar();
                JOptionPane.showMessageDialog(null, "============ IMPLEMENTACIÓN DE  COLAS
============"+"\n\n"+
                    "--- Se eliminó el elemento: "+elemento+" de la cola");
                mostrarCola(cola);
            }else{
                JOptionPane.showMessageDialog(null, "============ IMPLEMENTACIÓN DE  COLAS
============"+"\n\n"+
                    "La cola está vacía, No pueden quitar datos \n\n");
            }
            break;
        case 3:
            mostrarCola(cola);
            break;
        case 4:
            cola.limpiarCola();
            JOptionPane.showMessageDialog(null, "============ IMPLEMENTACIÓN DE COLAS
============"+"\n\n"+
                    "La cola está vacía, No pueden quitar datos \n\n");
            mostrarCola(cola);
            break;
        case 5:
            break;
        }
      }while(opcion != 5);
   }
}
```

EJERCICIOS/ACTIVIDADES

- La oficina de servicio al cliente de una entidad dedicada a los créditos educativos, presta diariamente el servicio a las personas que realizan préstamos para estudiar. La persona encargada de la recepción debe registrar el nombre y la identificación de la persona que llega para ser atendida. Se debe registrar en una cola de turnos la información para que el recepcionista tenga un listado en el orden que pasara a ser atendida cada persona.

  Después que la persona pase de ser atendida, será eliminada de la cola. Si una persona cuando se llame para ser atendida no se encuentra en la oficina, la persona perderá el turno y su información pasara al final de la cola de servicios.

  También se debe contar con una cola auxiliar en donde se almacene en orden de atención, la información de todas las personas atendidas durante el día, con el fin de imprimir un listado con esta información. Para la implementación en Java de la cola, la aplicación debe funcionar utilizando una lista enlazada o un vector como estructura auxiliar.

BIBLIOGRAFÍA

- Zahonero, I, y Joyanes Aguilar, L. (1999). Estructura de Datos - Algoritmos, Abstracción y Objetos. España: McGraw-Hill.
- Allen Weiss, M. (2004). Estructuras de datos en Java. España: Addison Wesley - Pearson. 776 pp.
- Cairó, O. & Guardati, S. (2002). Estructuras de datos (2nd ed.). México: McGraw-Hill.
- Joyanes, L. (2000). Programación en C++: Algoritmos, estructuras de datos y objetos (1st ed.). Madrid: McGraw-Hill.
- Kernighan, B., & Ritchie, D. (1991). El Lenguaje de Programación C (2nd ed.). México: Prentice Hall.
- Stroustrup, B. (2002). El Lenguaje de Programación C++ (1th ed.). Madrid: Pearson Educación.