9 | Cola en arreglo

Meta

Que el alumno domine el manejo de información almacenada en una *Cola*.

Objetivos

Al finalizar la práctica el alumno será capaz de:

- Implementar el tipo de dato abstracto *Cola* utilizando un arrelgo.

Antecedentes

Una `Cola` es una estructura de datos caracterizada por:

1. Ser una estructura de tipo `FIFO`, esto es que el primer elemento que entra es el primero que sale.
2. Tener un tamaño dinámico.
3. Ser lineal.

A continuación se define el tipo de dato abstracto `Cola`.

Definición 9.1: Cola

Una *Cola* es una estructura de datos tal que:

1. Tiene un número variable de elementos de tipo T.
2. Mantiene el orden de los datos ingresados, permitiendo únicamente el acceso al primero y el último.
3. Cuando se agrega un elemento, éste se coloca al final de la `Cola`.

4. Cuando se elimina un elemento, éste se saca del inicio de la `Cola`.

Nombre: $\mathtt{Vector}$.
Valores: $\mathbb{N}$, T, con $\mathtt{null} \in$ T.
Operaciones:

Constructores :

Cola() : $\emptyset \to \mathtt{Cola}$
Precondiciones : $\emptyset$
Postcondiciones : Una $\mathtt{Cola}$ vacía.

Métodos de acceso :

mira(this) $\to$ e: $\mathtt{Cola} \to$ T
Precondiciones : $\emptyset$
Postcondiciones :
- $e \in$ T, e es el elemento almacenado al inicio de la $\mathtt{Cola}$.

Métodos de manipulación :

forma(this, e) : $\mathtt{Cola} \times \mathrm{T} \xrightarrow{?} \emptyset$
Precondiciones : $\emptyset$
Postcondiciones :
- El elemento e es asignado al final de la $\mathtt{Cola}$.

atiende(this) $\to$ e: $\mathtt{Cola} \to$ T
Precondiciones :$\emptyset$
Postcondiciones :
- Elimina y devuelve el elemento e que se encuentra al inicio de la $\mathtt{Cola}$.

La interfaz `ICola` que debes implementar, contiene estos métodos. La única diferencia con dicha descipción radica en el constructor debido a la forma en que serán almacenados los datos y lo describiremos más adelante.

Para programar una cola en un arreglo, es necesario utilizar algunos trucos:

1. Se debe tener dos enteros indicando las posiciones del primer elemento (cabeza) y último elemento en la cola.
2. Los elementos se colocan a la derecha de la cabeza, módulo la longitud del arreglo, siempre que haya espacios disponibles. Si no hay espacios, se debe cambiar el arreglo por uno más grande.
3. Los elementos se remueven de la posición de la cabeza y el inidicador de esta se recorre a la casilla siguiente, a la derecha, módulo la longitud del arreglo. Un

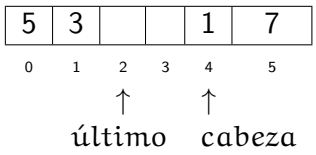

Figura 9.1 Cola en un arreglo, cuando ya se han eliminado elementos de la cabeza y se han formado elementos más allá de la longitud del buffer.

ejemplo se muestra en la Figura 9.1.

Desarrollo

En esta práctica se implementará la Cola utilizando arreglos. De nuevo se hará una extención de la clase `ColeccionAbstracta`. Por razones didácticas, no se permite el uso de ninguna clase que se encuentre en el API de Java, por ejemplo clases como `Vector`, `ArrayList` o cualquiera que haga el manejo de arreglos dinámicos. Excepción a esto es `java.util.Arrays` de la cual puedes usar el método `copyOf`. Fuera de esta excepción, del API de Java puedes importar únicamente interfaces y excepciones.

1. Crea un constructor que reciba como parámetro un arreglo de tamaño cero, del mismo tipo que la clase.
   Si el arreglo no es de tamaño cero lanza una `IllegalArgumentException`. Utilizarás este arreglo para crear el buffer en forma genérica. Utiliza la fución `Arrays.copyOf()` del API de `Java`para crear el arreglo. También debe recibir un tamaño inicial para el buffer de tipo entero. El encabezado de tu constructor quedará:

```
1 public ColaArreglo(E[] a, int tamInicial);
```

2. Crea otro constructor que sólo reciba el arreglo. Asignarás un valor inicial para el buffer con un tamaño por defecto que tú puedes elegir. Puedes llamar a tu otro constructor para no repetir el trabajo:

```
1 public ColaArreglo(E[] a) {
2   this(a, DEFAULT_INITIAL_SIZE);
3 }
```

3. Programa el método para agregar un elemento. Ojo: en este caso las dimensiones del arreglo no cambian si se llega al final, es posible que haya espacios vacíos al inicio del arreglo y deberás reutilizarlos antes que cambiar el tamaño del arreglo. Así puedes ahorrar tiempo, al no copiar a todos al nuevo arreglo. Verifica que funcione.

4. Programa el método para sacar un elemento. Deberás ir recorriendo la cabeza según sea necesario, dejando y hueco a su izquierda. Verifica que funcione.

5. Continúa con los otros métodos de la interfaz ICola, así como los necesarios para que tu clase compile. Puedes hacer que tu clase extienda a ColeccionAbstracta<E> de la práctica pasada para ahorrar un poco de trabajo. Recuerda que, dado que ICola extiende Collection<E>, debes implementar tu propio iterador para `ColaArreglo` con sus métodos correspondientes.

6. Un caso particular ocurrirá con los métodos `remove, removeAll` y `retainAll`. Recuerda que siguiendo la definición del tipo de datos abstracto, en una cola sólo tenemos acceso al primero y al último elemento. Por lo tanto no podemos eliminar elementos arbitrariamente.
   Sobrecarga estos métodos y lanza `UnsupportedOperationException`.

Preguntas

1. ¿Qué método utilizas para detectar cuando la cola está vacía?

2. ¿Qué fórmula utilizas para detectar cuando el buffer de la cola está lleno?

3. ¿Cuál es la complejidad para el mejor y peor caso de los métodos `mira`, `forma` y `atiende`? Justifica.