9 | TDA Cola

Definición

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

1. Ser una estructura de tipo FIFO[1], 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: Cola.
Valores: T, con $\mathrm{null} \notin$ T y $\mathbb{B}$.
Operaciones: Sea this la cola sobre la cual se está operando.

Constructores:

Cola(): $\emptyset \to \mathrm{Cola}$
Precondiciones: $\emptyset$
Postcondiciones:
- Se tiene una Cola vacía.

[1]Por sus siglas en inglés: *First In First Out*.

Métodos de acceso :

vacía?(this) → b: $\mathtt{Pila} \to \mathbb{B}$
Precondiciones: $\emptyset$
Postcondiciones:
- $\mathtt{b} \in \mathbb{B}$, $\mathtt{b} = \mathtt{true}$ si no hay elementos almacenados en la cola, $\mathtt{b} = \mathtt{false}$ en caso contrario.

mira(this) → e: $\mathtt{Cola} \to \mathsf{T}$
Precondiciones : $\emptyset$
Postcondiciones :
- $e = \mathtt{null}$ si la cola está vacía.
- $e \in \mathsf{T}$, e es el elemento almacenado al inicio de la cola si no está vacía.

Métodos de manipulación:

forma(this, e): $\mathtt{Cola} \times \mathsf{T} \xrightarrow{?} \emptyset$
Precondiciones: $e \neq \mathtt{null}$
Postcondiciones: Sea $\hat{e}$ el elemento que estaba al final de la cola.
- El elemento e es asignado al final de la cola.
- $\hat{e}$ antecede a e.

atiende(this) → e: $\mathtt{Cola} \to (\mathsf{T} \cup \mathtt{null})$
Precondiciones: $\emptyset$
Postcondiciones:
- Si la cola está vacía devuelve $\mathtt{null}$.
- Si la cola no está vacía, sea e el elemento al frente de la cola:
  - ⋆ Si e tenía un elemento $\hat{e}$ detrás de él, $\hat{e}$ queda al frente de la cola, si no la cola queda vacía.
  - ⋆ e ya no está almacenado en la cola.
  - ⋆ Devuelve el elemento e.

La interfaz `Cola`, 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 y en el método `vacía`, que no puede llevar el signo de interrogación.

Actividad 9.1

Revisa la documentación de la clase Queue de `Java`. ¿Cuáles serían los métodos equivalentes a los definidos aquí? ¿En qué difieren?

---

11 | 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

Arreglo

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

1. Se deben tener dos enteros indicando las posiciones del primer elemento (cabeza) y último elemento en la cola.

2. Los elementos se colocan a la derecha del último elemento, módulo la longitud del arreglo, siempre que haya espacios disponibles. Si no hay espacios, se debe cambiar el arreglo por uno más grande. Al hacer este cambio la cabeza quedará en la posición cero del arreglo nuevo.

3. Los elementos se remueven de la posición de la cabeza y el indicador de esta se recorre a la casilla siguiente, a la derecha, módulo la longitud del arreglo. Un ejemplo se muestra en la Figura 11.1.

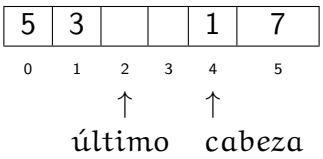

Figura 11.1 Cola en un arreglo, cuando ya se han eliminado elementos de la cabeza y se han formado elementos más allá de la longitud del buffer.

Iterador

La idea para implementar el iterador es semejante a la de la cola en arreglo, con las siguientes diferencias:

- El atributo que indica el siguiente elemento a devolver se inicializa en la cabeza de la cola, cuando el iterador es creado, cuidado porque ahora este valor no siempre es cero.
- Al recorrerse al elemento siguiente el indicador se incrementa en uno módulo la longitud del arreglo, pues el elemento siguiente al que está almacenado en la última casilla del arreglo se encontraría en la casilla cero.
- Para determinar si hay un elemento siguiente debe considerarse que el final de la cola puede estar antes o después de la cabeza. Hay varias formas de hacerlo, una es comparar el índice siguiente con el índice de la cabeza.

Desarrollo

En esta práctica se implementará la Cola utilizando arreglos. De nuevo se hará una extención de la clase `ColecciónAbstracta<E>`. Por razones didácticas, no se permite el uso de ninguna clase que se encuentre en el API de Java, por ejemplo clases como `Vector<E>`, `ArrayList<E>` 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 la clase `ColaArreglo<E>` en el paquete `ed.estructuras.lineales`, que implemente la interfaz `Cola<E>` y extienda `ColecciónAbstracta<E>`. Agrega los atributos necesarios.
2. 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 `IllegalSizeException`. 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);
```

3. Crea otro constructor que sólo reciba el arreglo. Asignarás un valor inicial para el buffer con un tamaño por defecto que tú puedes elegir; como se trata de un valor ligeramente arbitrario, la usanza es crear un atributo `final static`, es decir, una constante con el valor que hayas elegido de modo que, si luego quieres cambiar el valor, tu código no se vea afectado[1]. Puedes llamar a tu otro constructor para no repetir el trabajo:

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

4. Programa el método `mira` suponiendo que la cola puede estar vacía o ya tener elementos.

5. Programa el método `forma` 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.

6. Programa el método `atiende` para sacar un elemento. Deberás ir recorriendo la cabeza según sea necesario, dejando el hueco a su izquierda módulo la longitud del arreglo. Verifica que funcione.

7. Agrega los métodos faltantes de Collection<E>:

   - `public Iterator<E> iterator()`
   - `public boolean add(E e)`, será sinónimo de `forma` y siempre devuelve `true`.
   - `public void clear()` Es necesario sobreescribir este método, porque tu iterador no tendrá operación `remove()`, así que la implementación de la superclase no funcionará. Tendrás que borrar todos los elementos en el buffer, siguiendo la filosofía de `Java` no dejes basura en él (objetos en el buffer que ya no están en la cola). Hay varias formas de hacerlo, puedes hacerlo a mano o usar el método `atiende` aunque no hagas nada con el objeto devuelto; no vayas a recorrer todo el arreglo, sólo las casillas ocupadas.

[1] Por convención los nombres de las constantes se escriben con mayúsculas y las palabras se separan con un guión bajo.

- `public boolean remove(Object o)` Compara `o` sólo con el objeto devuelto por `mira()`, si son iguales lo remueve, si no devuelve `false`. Si `o` es `null` lanza `NullPointerException`.

Dado que en una la no operaremos sobre elementos que no están en el tope, los métodos siguientes deberán ser sobreescritos para que lancen `UnsupportedOperationException`:

- `public boolean retainAll(Collection<?> c)`
- `public boolean removeAll(Collection<?> c)`

8. Implementa el iterador. Aunque en una cola sólo se pueden agregar y remover elementos en un extremo, necesitaremos un iterador que permitar ver todos los elementos en la cola, desde la cabeza hasta el último. Para esta práctica sólo programarás un constructor, que inicialice el iterador en al frente de la cola, y los métodos `next` y `hasNext` del iterador.

Preguntas

1. ¿Qué técnica 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.