Práctica 7
Cola con referencias

Estructuras de datos

1. Meta

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

2. Objetivos

Al finalizar la práctica el alumno será capaz de:

- Implementar el tipo de dato abstracto **Cola** utilizando nodos y referencias.

3. 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 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, este se coloca al final de la `Cola`.
4. Cuando se elimina un elemento, este se saca del inicio de la `Cola`.

Nombre: $Vector$.
Valores: $\mathbb{N}$, T, con $null \in T$.
Constructores :
Cola() : $\emptyset \rightarrow Cola$
Precondiciones : $\emptyset$
Postcondiciones : Una $Cola$ vacía.

Métodos de acceso :

mira(this) $\rightarrow$ **e:** $Cola \rightarrow T$

Precondiciones : $\emptyset$

Postcondiciones :

- $e \in T$, e es el elemento almacenado al inicio de la $Cola$.

Métodos de manipulación :

encolar(this, e) : $Cola \times T \xrightarrow{?} \emptyset$

Precondiciones : $\emptyset$

Postcondiciones :

- El elemento e es asignado al final de la $Cola$.

desencolar(this) $\rightarrow$ **e:** $Cola \rightarrow T$

Precondiciones :$\emptyset$

Postcondiciones :

- Elimina y devuelve el elemento e que se encuentra al inicio de la $Cola$.

A continuación se muestra la interfaz `Cola`

```java
/*
 * Código utilizado para el curso de Estructuras de Datos.
 * Se permite consultarlo para fines didácticos en forma personal.
 */
package estructuras.lineales;

import java.util.Collection;
import java.util.NoSuchElementException;

public interface Cola<E> extends Collection<E> {

    /**
     * Muestra el elemento al inicio de la Cola
     * @return Una referencia al elemento siguiente.
     * @throws NoSuchElemntException si la Cola esta vacía.
     */
    public E mira() throws NoSuchElementException;

    /**
     * Devuelve el elemento al inicio de la Cola y lo elimina.
     * @return Una referencia al elemento siguiente.
     * @throws NoSuchElementException si la Cola esta vacía.
     */
    public E desencolar() throws NoSuchElementException;

    /**
     * Agrega un elemento al final de la Cola.
     * @param e Referencia al elemento a agregar.
     */
    public void encolar(E e);

}
```

Actividad 1

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

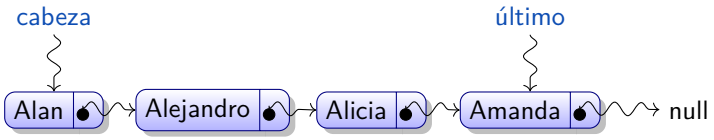

Figura 1: Representación en memoria de una cola, utilizando nodos y referencias.

4. Desarrollo

Se harán dos implementaciones para el TDA **Cola**: una con referencias y otra con arreglos. En esta práctica se realizará la implementación con referencias. De nuevo se hará una extención de la clase `ColeccionAbstracta`.

La implementación es análoga a la utilizada para la pila con referencias, con la diferencia de que necesitarás un atributo más en la clase cola: una referencia al último elemento de la estructura, pues ahí se formarán los nodos con los elementos entrantes (Figura 1).

1. Programa la clase `Nodo`. Si en la práctica sobre pilas la creaste dentro del paquete `estructuras.lineales`, puedes utilizar la que ya tienes. Si elegiste es programarlo como una clase estática interna de `PilaLigada`, puedes copiar y pegar el código dentro de tu clase `ColaLigada` y funcionará igual.
2. Programa la clase `ColaLigada`. No olvides implementar la interfaz `Cola`. Inicia con los métodos básicos y luego agrega el iterador que requiere `Collection`.

5. Preguntas

1. Explica cómo funcionan los métodos `encolar` y `desencolar`.
2. ¿Cuál es la complejidad de los métodos `mira`, `encolar` y `desencolar`? Justifica tus respuestas.

6. Forma de entrega

- Asegúrense de borrar todos los archivos generados, incluyendo archivos de respaldo usando `ant clean`.
- Copien el directorio `Practica7` dentro de un directorio `<apellido1_apellido2>` donde `apellido1` es el primer apellido de un miembro del equipo y `apellido2` es el primer apellido del otro miembro. Por ejemplo: `marquez_vazquez`.
- Borren el directorio libs en la copia.
- Agreguen un archivo `reporte.pdf` dentro del directorio `Practica7` de la copia, con el nombre completo de los integrantes del equipo, las repuestas a las preguntas de la sección anterior y cualquier comentario que quieran hacer sobre la práctica.
- Compriman el directorio `<apellido1_apellido2>` en un archivo `<apellido1_apellido2>.tar.gz`. Por ejemplo: `marquez_vazquez.tar.gz`.
- Suban este archivo en la sección correspondiente del aula virtual. Basta una entrega por equipo.

7. Forma de evaluación

Para su calificación final se tomarán en cuenta los aspectos siguientes:

- 70 % Calificación generada por las pruebas automáticas. Usaremos nuestros archivos, por lo que si realizan modificaciones a sus copias, éstas no tendrán efecto al momento de calificar.
- 15 % Documentación completa y adecuada. Entrega en el formato requerido, sin archivos `.class`, respaldos, bibliotecas de `JUnit` u otros no requeridos.
- 15 % Revisión manual del código, para verificar que se cumpla con las especificaciones, que no se haya copiado, etcétera.