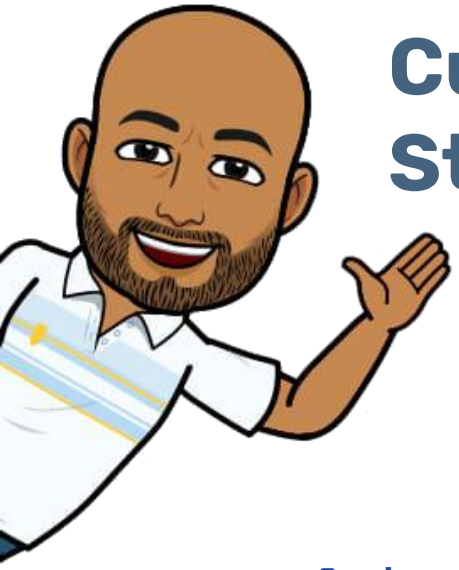


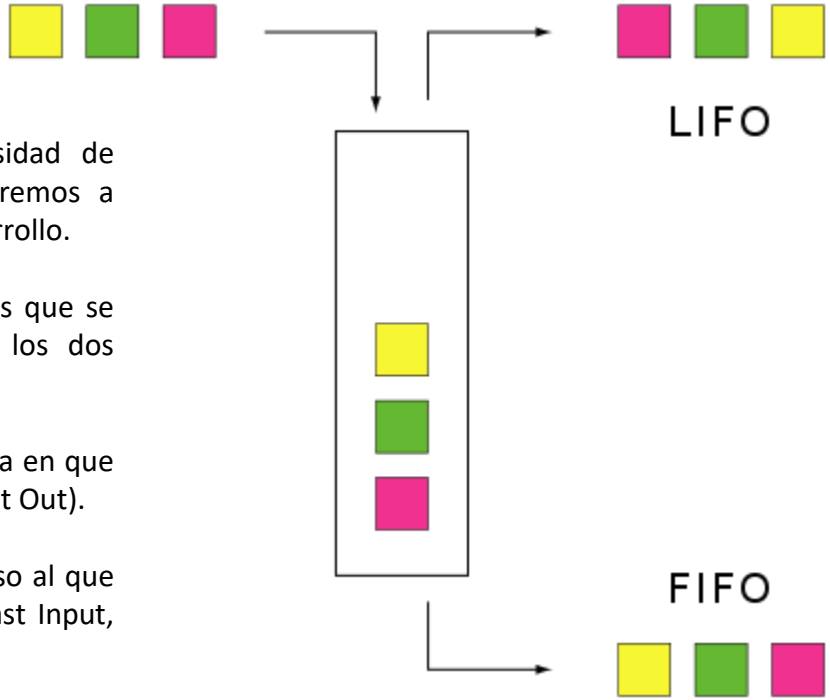
Curso de Java Standard

Ing. Octavio Robleto

octavio.robleto@gmail.com

https://octaviorobleto.com

Introducción



En el desarrollo de aplicaciones se nos puede presentar la necesidad de implementar un sistema de colas y/o pilas, las estructuras que veremos a continuación nos proporcionan métodos para facilitar el proceso de desarrollo.

Las Colas y las Pilas representan una estructura lineal de objetos en los que se pueden agregar o eliminar los elementos únicamente por uno de los dos extremos.

Los elementos de una cola se eliminan de la colección en la misma forma en que fueron insertados, a esta estructura se les conoce como FIFO (First In, First Out).

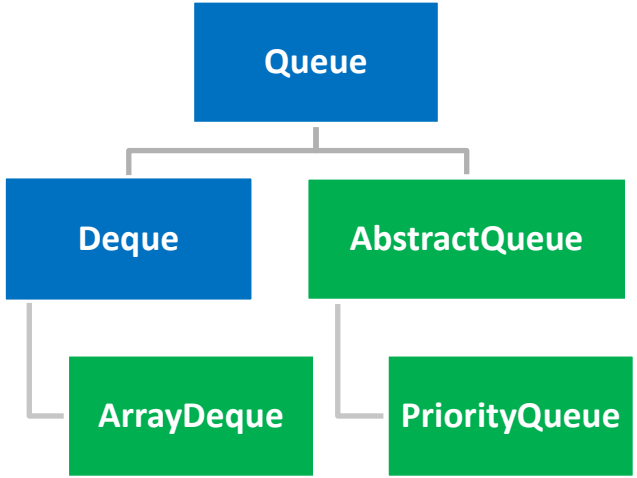
Los elementos de una pila se eliminan de la colección en el orden inverso al que se insertaron, a esta estructura se les conoce como estructura LIFO (Last Input, First Out).

Interfaz Queue

La interfaz **Queue** tiene una peculiaridad y es que ofrece métodos para poder trabajar con colas (Queue en Ingles).

La clase **AbstractQueue** Proporciona una implementación esquelética de la interfaz **Queue** y simplemente agrega implementaciones para los métodos **equals** y **hashCode**.

La interfaz **Deque (double ended queue)** o cola de dos extremos proporciona a sus implementaciones métodos para la eliminación e inserción de elementos tanto al comienzo como al final de las colas, la colección **LinkedList** también implementa esta interfaz.



Cabe destacar que estas colecciones no permiten agregar (add) elementos nulos.

Métodos Queue

Adicionalmente de los métodos que nos proporciona la interfaz **Collection, Queue** añade unos nuevos:

Tipo	Método	Descripción
E	element()	Devuelve el elemento que se encuentre al principio de la cola
boolean	offer(E e)	Inserta un elemento al final de la cola
E	peek()	Elimina el elemento que se encuentre al principio de la cola
E	poll()	Elimina el elemento que se encuentre al principio de la cola

```
E elemento1, elemento2, elemento3;
Coleccion<E> coleccion = new Implementacion<>();

coleccion.add(elemento1);
coleccion.add(elemento2);

// devuelve el elemento al principio de la cola
System.out.println(coleccion.element());

// Agrega el elemento al principio de la cola
coleccion.offer(elemento3);

while(!coleccion.isEmpty()) {
    // Devuelve el elemento al principio de la cola
    System.out.println(coleccion.peek());

    // Devuelve y remueve el elemento que se encuentre al principio de la cola
    System.out.println(coleccion.poll());
}
```

Con estos métodos no nos vemos en la necesidad de usar un iterador y la forma de resolver la eliminación automática de los objetos la hacemos con el bucle while hasta que la colección se encuentre vacía.

PriorityQueue

Este implementación almacena los elementos y los ordena según una prioridad dada, si no se le asigna prioridad asumirá que el orden de los objetos será por el orden natural.

```
// Clase que implementa Comparator
ComparadorPrioritario comparadorPrioritario =  new ComparadorPrioritario();

// Cola Prioritaria con orden natural
Queue<String> nombresA = new PriorityQueue<>();

// Cola Prioritaria con orden alternativo
PriorityQueue<String> nombresB = new PriorityQueue<>(comparadorPrioritario);
```

Métodos Deque

Adicionalmente de los métodos que nos proporciona la interfaz **Collection y Queue, Deque** añade unos nuevos:

Tipo	Método	Descripción
boolean	offerFirst(E e)	Inserta un elemento al principio de la cola
boolean	offerLast(E e)	Inserta un elemento al final de la cola
E	peekFirst()	Elimina el elemento que se encuentre al principio de la cola
E	pollFirst()	Elimina el elemento que se encuentre al principio de la cola
E	peekLast()	Elimina el elemento que se encuentre al final de la cola
E	pollLast()	Elimina el elemento que se encuentre al final de la cola

```
E elemento1, elemento2, elemento3, elemento4;
Coleccion<E> coleccion = new Implementacion<>();

coleccion.add(elemento1);
coleccion.add(elemento2);

// Agrega el elemento al principio de la cola
coleccion.offerFirst(elemento3);

// Agrega el elemento al principio de la cola
coleccion.offerLast(elemento4);

//Trabajar como cola
while(!coleccion.isEmpty()) {
    // Devuelve el elemento al principio de la cola
    System.out.println(coleccion.peekFirst());

    // Devuelve y remueve el elemento que se encuentre al principio de la cola
    System.out.println(coleccion.pollFirst());
}

//Trabajar como pila
while(!coleccion.isEmpty()) {
    // Devuelve el elemento al principio de la pila
    System.out.println(coleccion.peekLast());

    // Devuelve y remueve el elemento que se encuentre al principio de la pila
    System.out.println(coleccion.pollLast());
}
```

ArrayDeque

Este implementación almacena los elementos para trabajarlos como colas si se declaran como **Queue** o como colas y pilas si se declaran como **Deque** o **ArrayDeque**.

```
// como cola
Queue<String> nombresA = new ArrayDeque<>();

// como cola o pila
Deque<String> nombresB = new ArrayDeque<>();

// como cola o pila
ArrayDeque<String> nombresC = new ArrayDeque<>();
```

La colección LinkedList también implementa esta interfaz por lo que podemos utilizarla para el caso de Pilas y Colas si se declaran como Deque o LinkedList