educación IT
CODE YOUR FUTURE

Bootcamp Java Developer

Fase 1 - Java Analyst

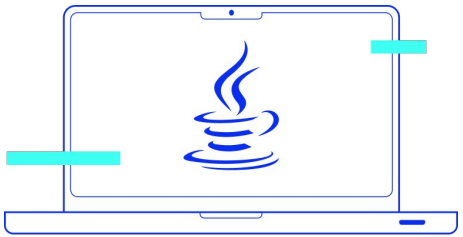
Módulo 9

Colecciones

Introducción

En el desarrollo de aplicaciones, se puede presentar la necesidad de implementar un **sistema de colas y/o pilas.** Las estructuras que se verán a continuación 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 desde uno de los dos extremos**.

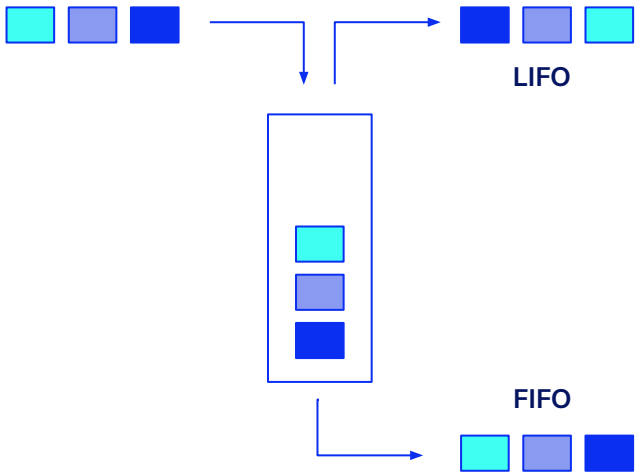


FIFO

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*).

LIFO

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

Interfaz Queue

La interfaz **Queue** tiene la peculiaridad de ofrecer métodos para poder trabajar con **colas** (*Queue* en inglés).

La clase AbstractQueue

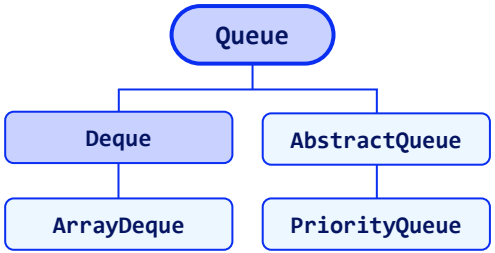
Cola abstracta

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

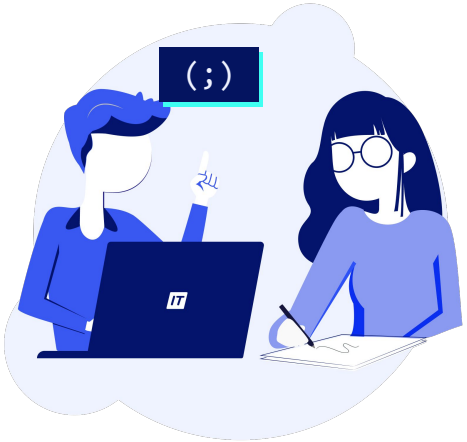
Proporciona a sus implementaciones métodos para la eliminación e inserción de elementos tanto al comienzo como al final de las colas. La colección `LinkedList` también implementa esta interfaz.



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.



Ejemplo

```
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());
}
```

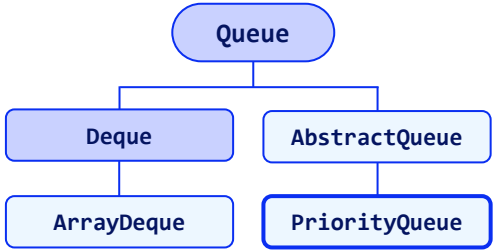
PriorityQueue

Esta 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 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);
```

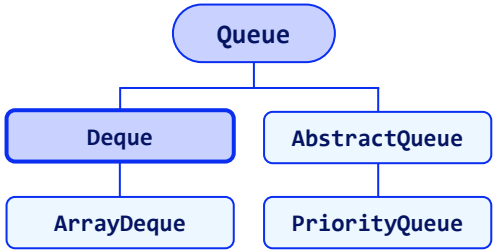


Interfaz Deque

Métodos Deque

Deque añade nuevos métodos a los que ya proporcionan las interfaces `Collection` y `Queue`:

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.



Ejemplo

```
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());
}
```

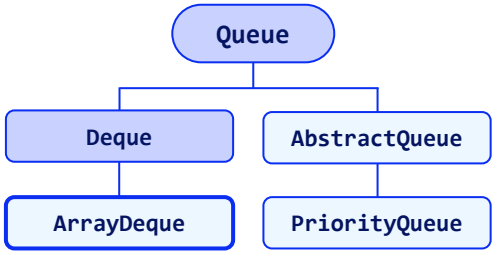
ArrayDeque

Esta 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<>();
```



¡Sigamos trabajando!