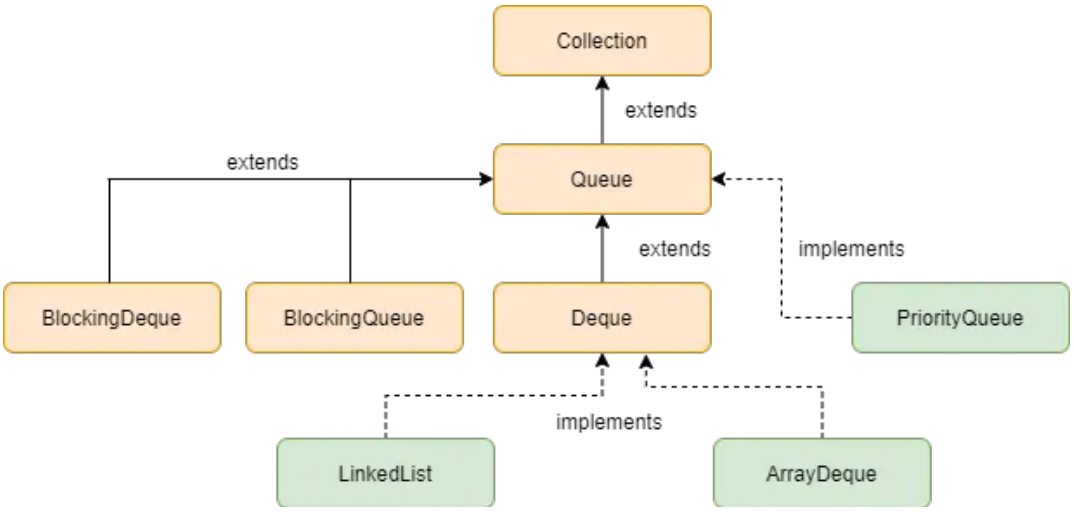
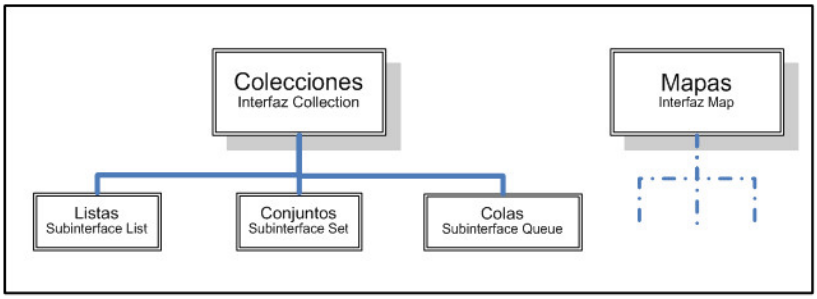
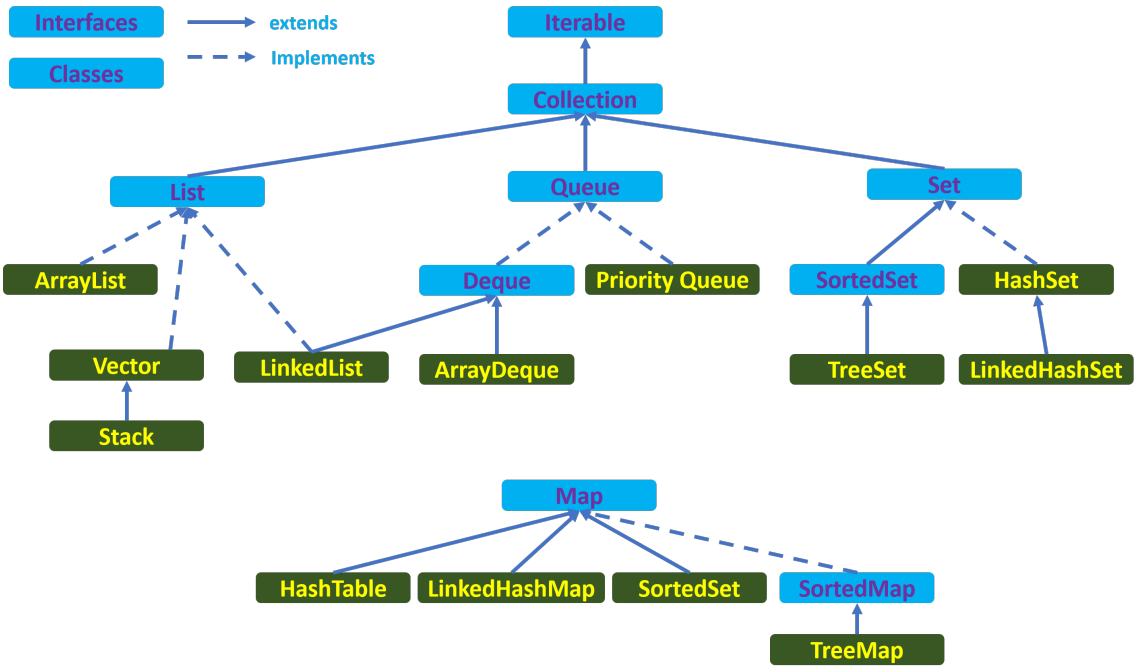
Colas

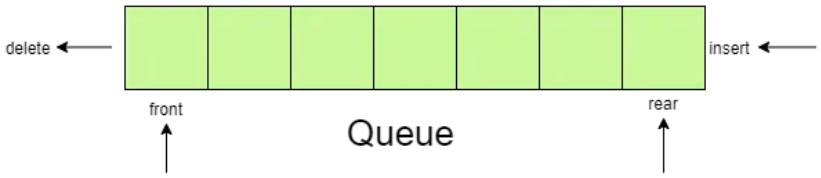

Collection
extends
extends
Queue
extends
implements
BlockingDeque
BlockingQueue
Deque
PriorityQueue
implements
LinkedList
ArrayDeque


Colecciones
Interfaz Collection
Mapas
Interfaz Map
Listas
Subinterface List
Conjuntos
Subinterface Set
Colas
Subinterface Queue


Interfaces
Classes
extends
Implements
Iterable
Collection
List
Queue
Set
ArrayList
Deque
Priority Queue
SortedSet
HashSet
Vector
LinkedList
ArrayDeque
TreeSet
LinkedHashSet
Stack
Map
HashTable
LinkedHashMap
SortedSet
SortedMap
TreeMap

Java Queue (interfaz)

- Agrega elementos por la parte trasera y los borra por la parte delantera
- Implementa el concepto de primero en entrar, primero en salir (FIFO), algunas colas se pueden utilizar también para implementar pilas (LIFO).
- Admite todos los métodos de la interfaz de colección
- Mantiene una colección ordenada de elementos



PriorityQueue (clase)

Es una clase que implementa la cola y procesa los elementos según una prioridad FIFO.

- **Prioridad por defecto (según el tipo de dato)**

Por ejemplo con String será alfabéticamente.

```
PriorityQueue<String> nombres= new PriorityQueue<>();

nombres.add("Paco");
nombres.add("Patricia");
nombres.add("Joe");
nombres.add("Juan Antonio");
nombres.add("Patri");

System.out.println(nombres); //El orden en el que se muestra es alfabéticamente

while (!nombres.isEmpty()) //Al sacar de la cola se verá la prioridad que hemos definido
    System.out.println(nombres.poll());
```

Salida por pantalla: [Joe, Juan Antonio, Paco, Patricia, Patri]
Joe
Juan Antonio
Paco
Patri
Patricia

- **Prioridad según clase comparadora**

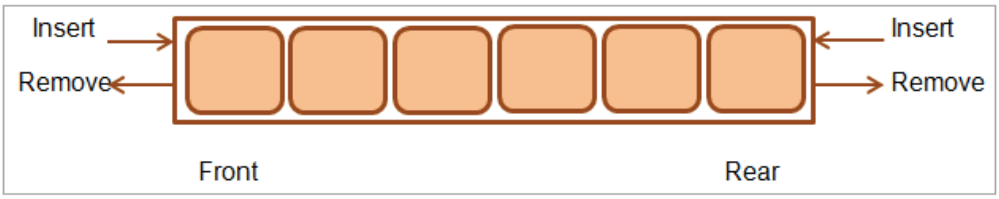
Acepta una clase comparadora en el constructor que determinará la prioridad de la cola.

```
//Creamos una cola donde la prioridad sea el tamaño del nombre
PriorityQueue<String> nombres= new PriorityQueue<>(new ComparaLongitud());

nombres.add("Paco");
nombres.add("Patricia");
nombres.add("Joe");
nombres.add("Juan Antonio");
nombres.add("Patri");

System.out.println(nombres); //El orden en el que se muestra no coincide con la entrada

while (!nombres.isEmpty()) //Al sacar de la cola se verá la prioridad que hemos definido
    System.out.println(nombres.poll());
```

Salida por pantalla: [Joe, Patri, Paco, Juan Antonio, Patricia]
Joe
Paco
Patri
Patricia
Juan Antonio

Deque (interfaz)

Es una interfaz que extiende la interfaz cola. La Deque o "**cola de dos extremos**" en Java es una estructura de datos en la que **podemos insertar o eliminar elementos de ambos extremos.**
Admite la implementación de la cola, que es el primero en entrar, el primero en salir (FIFO), y la implementación de la pila, que es el último en entrar, primero en salir (LIFO).



Inserción de elementos en un deque:

Arriba Abajo

Sin excepción	`offerFirst()`	`offer()`, `offerLast()`
Excepción	`addFirst()`, `push()`	`add()`, `addLast()`

Recuperar elementos de un deque:

Primer elemento (superior), sin eliminación Primer elemento (superior), eliminación

Sin excepción	`peek()`, `peekFirst()`	`poll()`, `pollFirst()`
Excepción	`getFirst()`, `element()`	`remove()`, `removeFirst()`, `pop()`

Último elemento (inferior), sin eliminación Último elemento (inferior), eliminación

Sin excepción	`peekLast()`	`pollLast()`
Excepción	`getLast()`	`removeLast()`

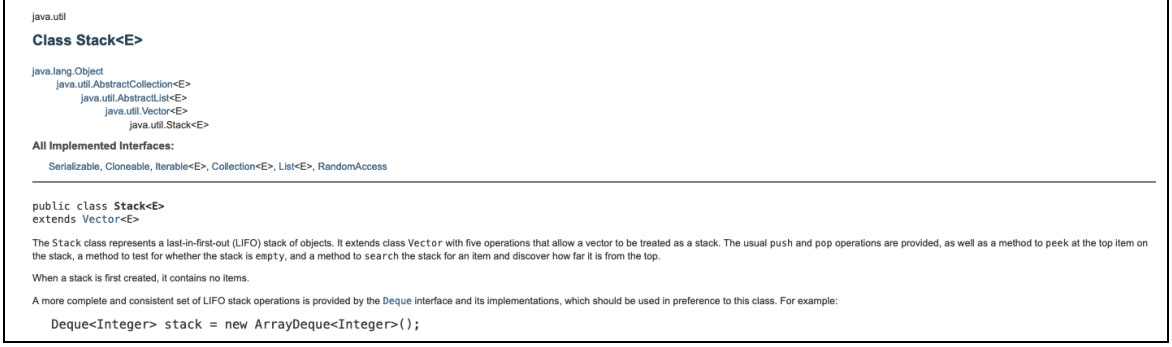
`add()`, `addFirst()` y `addLast()` lanzan excepción si la cola está llena.

`offer()` , `offerFirst()` y `offerLast()` devuelven true si el elemento se inserta con éxito; si la cola deque está llena, estos métodos regresan false.

ArrayDeque (clase)

La clase ArrayDeque implementa la interfaz Deque, lo que significa que podemos **insertar y eliminar los elementos de ambos lados**

Nota: Java tiene una clase para Pilas (Stack), ya en desuso. Según figura en el API de Java se recomienda para la implementación de pilas el uso de la interfaz Deque a partir de la implementación de ArrayDeque.



java.util

Class Stack<E>

java.lang.Object
  java.util.AbstractCollection<E>
    java.util.AbstractList<E>
      java.util.Vector<E>
        java.util.Stack<E>

All Implemented Interfaces:

Serializable, Cloneable, Iterable<E>, Collection<E>, List<E>, RandomAccess

---

public class **Stack<E>**
extends Vector<E>

The Stack class represents a last-in-first-out (LIFO) stack of objects. It extends class Vector with five operations that allow a vector to be treated as a stack. The usual push and pop operations are provided, as well as a method to peek at the top item on the stack, a method to test for whether the stack is empty, and a method to search the stack for an item and discover how far it is from the top.

When a stack is first created, it contains no items.

A more complete and consistent set of LIFO stack operations is provided by the Deque interface and its implementations, which should be used in preference to this class. For example:

```
Deque<Integer> stack = new ArrayDeque<Integer>();
```

No tiene restricciones de capacidad y crece según sea necesario para admitir el uso. No es seguro para subprocesos, lo que significa que, en ausencia de sincronización externa, ArrayDeque no admite el acceso simultáneo de varios subprocesos.

```java
import java.util.*;
public class ArrayDequeDemo {
    public static void main(String[] args)
    {
        // Initializing an deque
        Deque<Integer> de_que
            = new ArrayDeque<Integer>(10);

        // add() method to insert
        de_que.add(10);
        de_que.add(20);
        de_que.add(30);
        de_que.add(40);
        de_que.add(50);

        System.out.println(de_que);

        // clear() method
        de_que.clear();

        // addFirst() method to insert the
        // elements at the head
        de_que.addFirst(564);
        de_que.addFirst(291);

        // addLast() method to insert the
        // elements at the tail
        de_que.addLast(24);
        de_que.addLast(14);

        System.out.println(de_que);
    }
}
```

Ejecución:

```
[10, 20, 30, 40, 50]
[291, 564, 24, 14]
```

Insertando elementos en ArrayDeque

```java
public class PruebaColasDeque {

    public static void main(String[] args) {

         ArrayDeque<String> animales= new ArrayDeque<>();

        animales.add("Perro");

        animales.addFirst("Gato");

        animales.addLast("Caballo");

        System.out.println("ArrayDeque: " + animales);

    }
```

Console

<terminated> PruebaColasDeque [Java Application] /Users/macair/.p2/pool/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.macosx.x86_64_14.0.2.v20200815-0932/jre/bin/java (20 abr. 2023

```
ArrayDeque: [Gato, Perro, Caballo]
```

Recuperando elementos de una cola

```java
    public static void main(String[] args) {
         ArrayDeque<String> animales= new ArrayDeque<>();

        //Añade al final
        animales.add("Perro");

        //Añade al principio
        animales.addFirst("Gato");

        //Añade al final
        animales.add("León");

        //Añade al final
        animales.addLast("Caballo");

        System.out.println("ArrayDeque: " + animales);

        // Obtiene el primer elemento de la cola
        String firstElement = animales.getFirst();
        System.out.println("Primer Elemento: " + firstElement);
         // Obtiene el ultimo elemento
        String lastElement = animales.getLast();
        System.out.println("Último Elemento: " + lastElement);
```

Console

<terminated> PruebaColasDeque [Java Application] /Users/macair/.p2/pool/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.macosx.x86_64_14.0.2.v20200815-0932/jre/bin/java (20 abr. 2023

```
ArrayDeque: [Gato, Perro, León, Caballo]
Primer Elemento: Gato
Último Elemento: Caballo
```

Ejercicios propuestos:

Implementa un ArrayDeque que sirva para organizar los pedidos a domicilio de un restaurante (un FIFO)

Crea una menú con las siguientes opciones:
a. Nuevo pedido, se introduce en la cola
b. ¿Pedidos en cola?
c. ¿Primer pedido en cola?
d. Servir un pedido (mostrar numero de pedido, desaparecerá de la cola)

e. Pedido Vip
f. Salir del programa

Implementar una clase pila para añadir prendas de ropa para planchar, usando deque y con los siguientes métodos (un LIFO)

- insertar: inserta elemento en la pila, no devuelve nada. Como parámetro recibe solo el número a insertar.
- quitar: quita elemento de la pila, devuelve -1 si está vacía. No recibe parámetros
- vacia: informa si la pila está vacía o no, devuelve booleano. No recibe parámetros
- numelementos: dice cuantos elementos hay en la pila. No recibe parámetros

Después implementa un programa que haga uso de esa clase.