CODE (QUEUE)

CARATTERISTICHE GENERALI

- Comportamento definito FIFO
  - **First In, First Out**
  - primo inserito è il primo ad essere estratto
  - può essere ispezionato solo il primo oggetto della coda

UTILIZZO DELLE CODE

- Simulazione sportello bancario
  - clienti in coda
- File da stampare vengono inseriti in una coda di stampa
- Task scheduler

INTERFACCIA

- Definisce le operazioni:
  - enqueue: inserisce oggetto nella coda
  - dequeue: elimina dalla coda oggetto inserito per primo
  - getFront: esamina primo oggetto senza estrarlo
  - operazioni base del ADT "container"

```
public interface Queue extends Container {
    void enqueue(Object obj);
    Object dequeue();
    Object getFront();
}
```

REALIZZAZIONE DELLA CODA

- Definiamo due nuove **eccezioni**
  - `class EmptyQueueException extends RuntimeException`
  - `class FullQueueException extends RuntimeException`

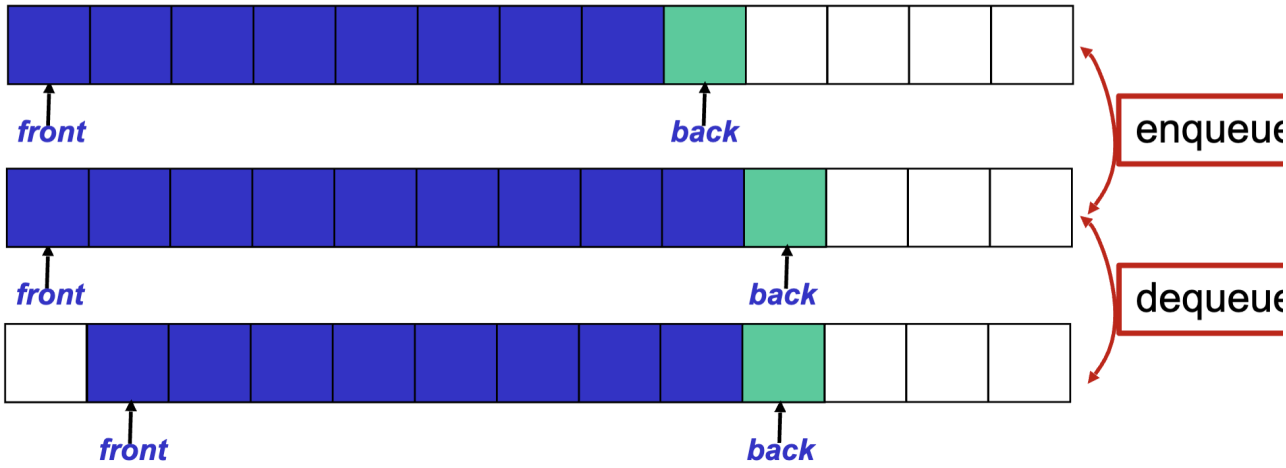
IMPLEMENTAZIONE CON ARRAY RIEMPITO SOLO IN PARTE

```java
public class SlowFixedArrayQueue implements Queue {
    protected Object[] array;
    protected int arraySize;
    public static final int INIT_SIZE = 100;

    public SlowFixedArrayQueue() {
        array = new Object[INIT_SIZE];
        makeEmpty();
    }

    public boolean isEmpty() {
        return arraySize == 0;
    }

    public void makeEmpty() {
        arraySize = 0;
    }

    public void enqueue(Object obj) {
        if (arraySize == array.length)
            throw new FullQueueException();
        array[arraySize++] = obj;
    }

    public Object dequeue() {
        Object obj = getFront();
        arraySize--;
        for (int i = 0; i < arraySize; i++) {
            array[i] = array[i + 1];
        }
        return obj;
    }

    public Object getFront() {
        if (isEmpty())
            throw new EmptyQueueException();
        return array[0];
    }
}
```

- Il metodo `dequeue` ha complessità $O(n)$
  - non è molto efficiente

IMPLEMENTAZIONE CON "ARRAY RIEMPITO SOLO NELLA PARTE CENTRALE"

- Si usano due indici
  - front: indica primo elemento nella coda
  - back: indica primo posto libero dopo ultimo elemento nella coda
  - numeri di elementi: back - front




```java
public class FixedArrayQueue implements Queue {
    protected Object[] array;
    protected int front, back;
    public static final int INIT_SIZE = 100;

    public FixedArrayQueue() {
        array = new Object[INIT_SIZE];
        makeEmpty();
    }

    public boolean isEmpty() {
        return back == front;
    }

    public void makeEmpty() {
        back = front = 0;
    }

    public void enqueue(Object obj) {
        if (back == array.length)
            throw new FullQueueException();
        array[back++] = obj;
    }

    public Object dequeue() {
        Object obj = getFront();
        front++;
        return obj;
    }

    public Object getFront() {
        if (isEmpty())
            throw new EmptyQueueException();
        return array[front];
    }
}
```

- Il metodo `dequeue` ha ora efficienza $O(1)$

IMPLEMENTAZIONE CON ARRAY A RIDIMENSIONAMENTO DINAMICO

```
public void enqueue(Object obj) {
    if (back == array.length)
        array = resize(2 * array.length);
    array[back++] = obj;
}
```

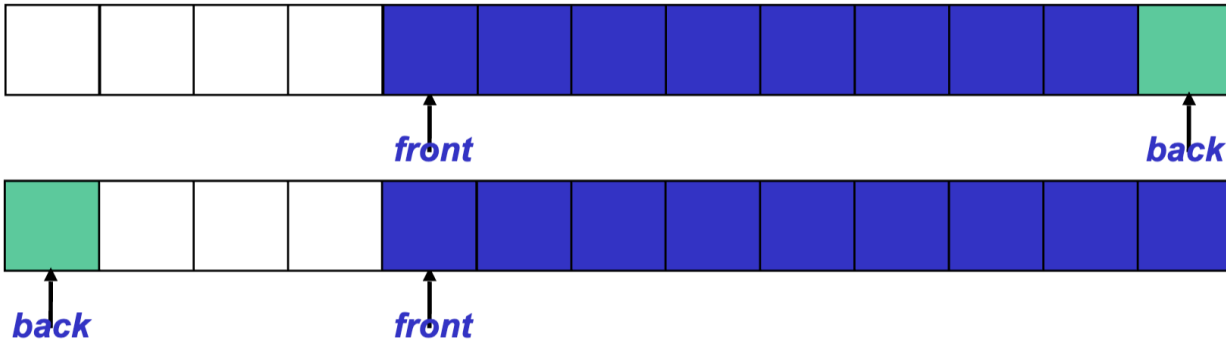
- Tutte le operazioni continuano ad avere efficienza: $O(1)$

> **Attenzione**
> Si rischia, se aumenta molto indice front, di avere un array quasi totalmente vuoto

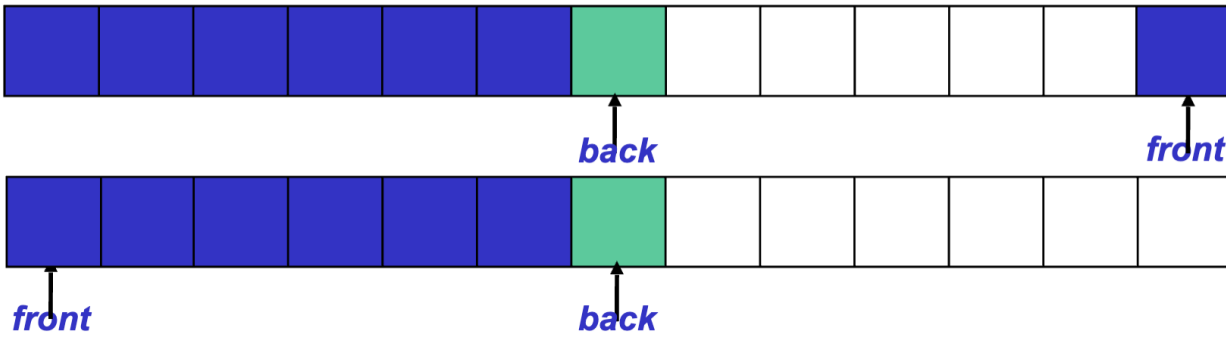
CODA CON ARRAY CIRCOLARE

- Array in cui, quando è pieno, si re-inizia ad inserire elementi dal primo elemento
- È pieno solo quando `front == back`

- Incremento dell'indice ***back*** da **n-1** a **0**



- Incremento dell'indice ***front*** da **n-1** a **0**

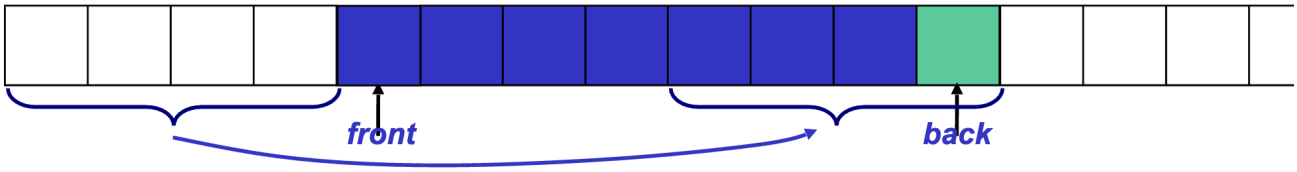


IMPLEMENTAZIONE CON ARRAY STATICI

```java
public class FixedCircularArrayQueue extends FixedArrayQueue {
    protected int increment(int index) {
        return (index + 1) % array.length;
    }

    public void enqueue(Object obj) {
        if (increment(back) == front)
            throw new FullQueueException();
        array[back] = obj;
        back = increment(back);
    }

    public Object dequeue() {
        Object obj = getFront();
        front = increment(front);
        return obj;
    }
}
```

RIDIMENSIONARE CON ARRAY CIRCOLARE

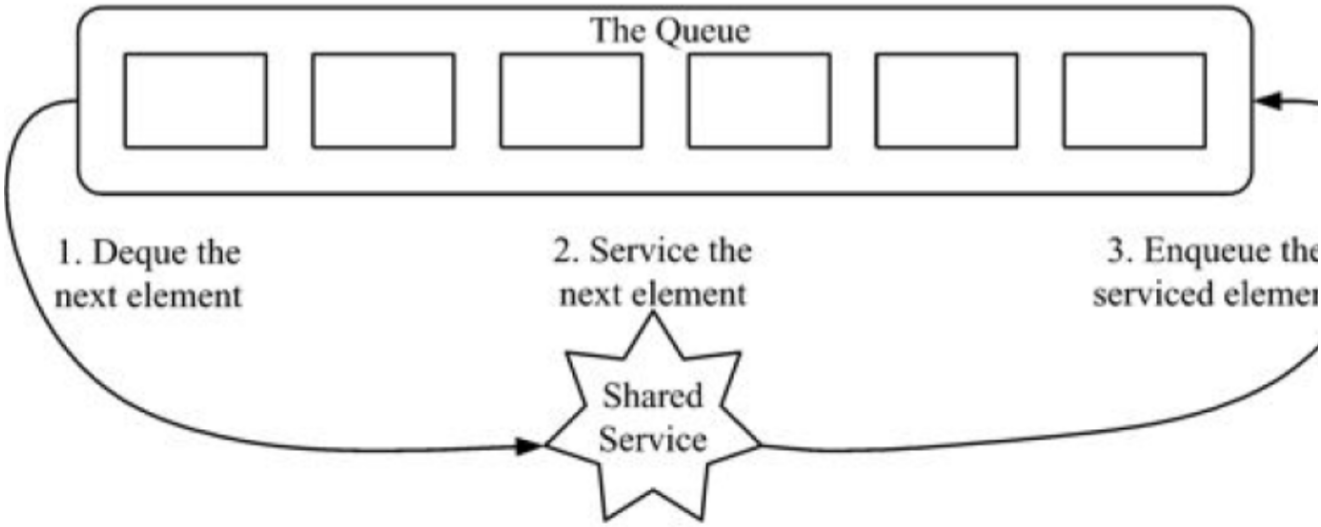
- Potrebbe dare array pieno anche quando l'indice `back` non è l'ultimo elemento
- Sposto gli elementi prima di front nella seconda metà dell'array



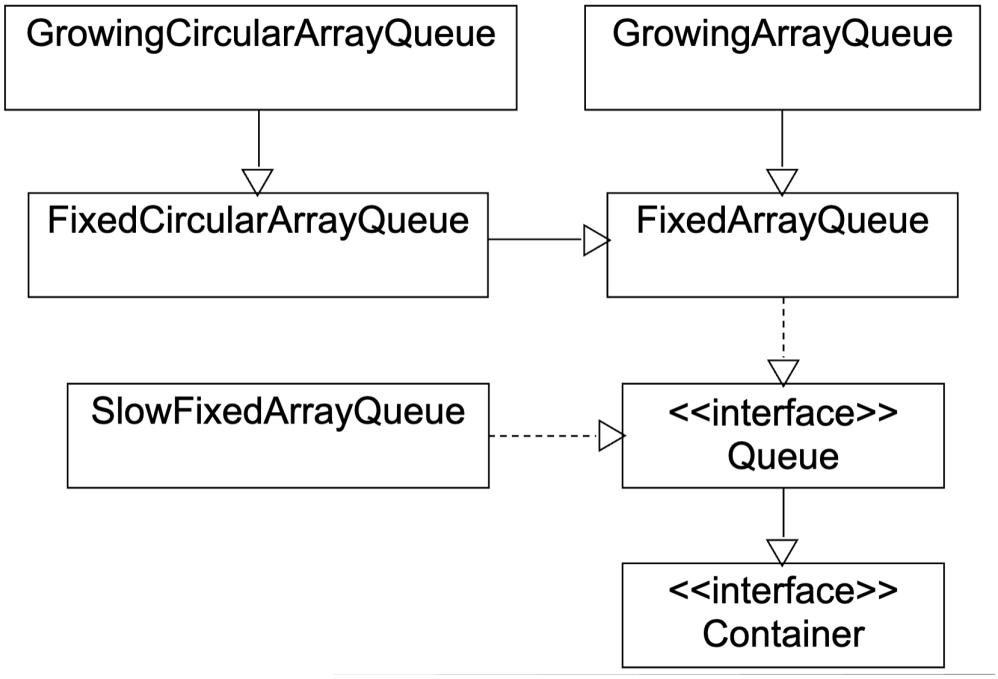

```java
public void enqueue(Object obj) {
    if (increment(back) == front) {
        array = resize(2 * array.length);
        if (back < front) {
            System.arraycopy(array, 0, array, array.length / 2, back);
            back += array.length / 2;
        }
    }
    array[back] = obj;
    back = increment(back);
}
```

CODA CIRCOLARE

- Politica round robin
  - la CPU esegue una porzione del processo che ha atteso più a lungo e poi viene re-inserito nell'ultima posizione



GERARCHIA DI CLASSI E INTERFACCE



CODA DOPPIA (DEQUE)

- Elementi possono venire inseriti ed estratti ai due estremi

INTERFACCIA

```
public interface Deque extends Container {
    void addFirst(Object obj);
    void addLast(Object obj);

    Object removeFirst();
    Object removeLast();

    Object getFirst();
    Object getLast();

    int size();
}
```

- Può essere realizzata con **array circolare ridimensionabile**
- Tutti i metodi hanno prestazioni $O(1)$

✓ **è da livello esame**