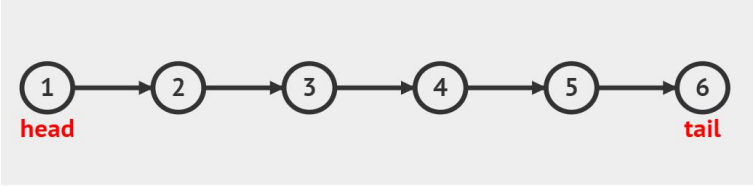
[6] Queue

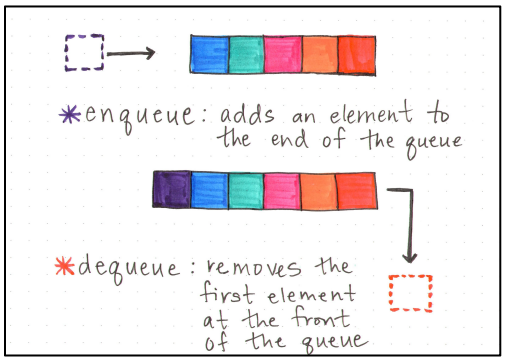
- Queue adalah struktur data linear yang menggunakan konsep FIFO (First In First Out). Contoh sederhana dari queue adalah antrian beli makan di KFC dimana pembeli yang datang duluan di layani lebih dulu .
- Kita hanya bisa menambah item baru pada ujung yang satu (tail/ekor) dan menghapusnya dari ujung yang lain (kepala/head).

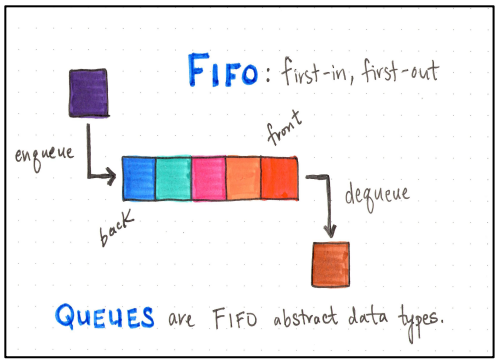
Pada Queue terdapat satu buah pintu masuk di ujung belakang dan satu buah pintu keluar di depannya..

To visualize a queue, think of people lining up.



<u>Ada beberapa istilah dalam penggunaan queue:</u>





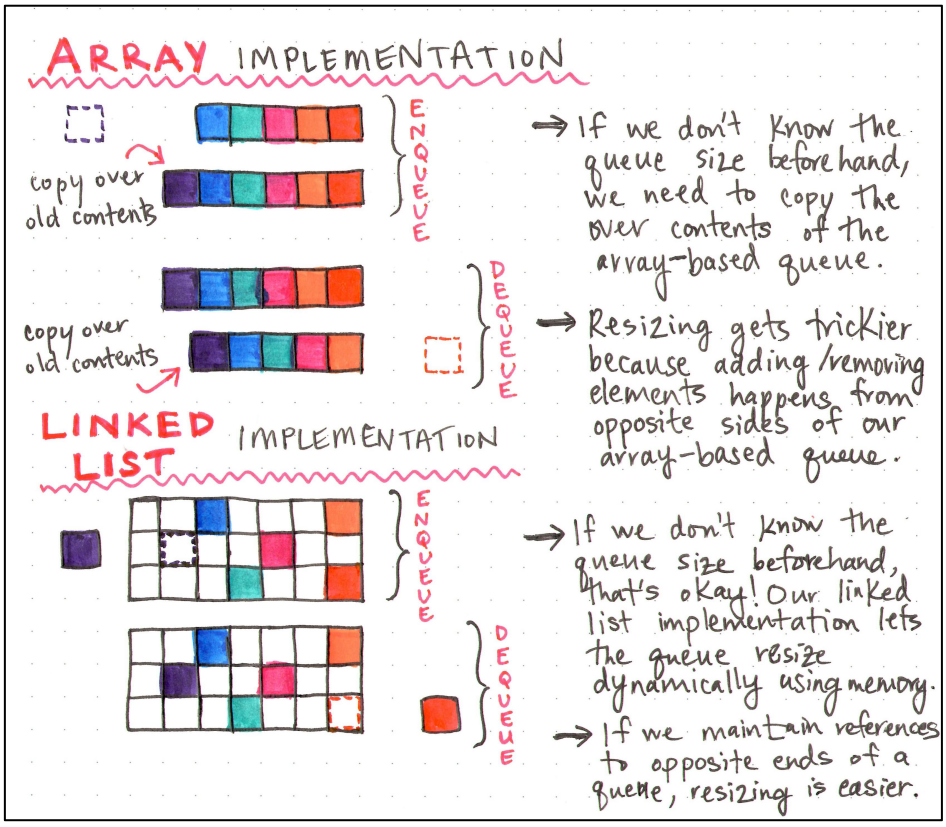
- **Enqueue** -> menambah item ke queue
- **Dequeue** -> menghapus item dari queue
- Dan yang paling penting, ketika mengimplementasikan queue kita harus tau mana **head (front)** dan **tail (rear)** nya,

Kapan queue digunakan?

Ketika kita ingin menjalankan proses berdasarkan antriannya, misal:

- Breadth First Search (salah satu metode pencarian di struktur data tree)
- Kondisi dimana resource dibagikan pada banyak pemakai -> CPU scheduling, disk scheduling, job scheduling

Impelementasi Queue menggunakan Array vs Linked List



Operasi Queue dengan memakai Java Collection Framework

1. Mendeklarasikan package yang ada di java

```
// Mengimport satu per satu
import java.util.Queue;
import java.util.LinkedList;
import java.util.Iterator;

// Mengimport sekaligus
Import java.util.*;
```

2. Menginstansiasi objek Queue

```
// Tipe Data Spesifik
Queue<String> queue = new LinkedList<>();

// Tipe Data Dinamis
Queue queue = new LinkedList();
```

3. Memasukkan data ke dalam objek queue (Enqueue)

```
queue.add("senin");
queue.add("selasa");
queue.add("rabu");
```

4. Mencetak isi queue

```
//menggunakan iterator
Iterator iterator = queue.iterator();
While(iterator.hasNext()){
    String text = iterator.next();
    System.out.println(text);
}

//langsung
System.out.println(queue);
```

5. Mengecek queue Kosong atau Tidak

```
// isEmpty() akan bernilai true jika queue kosong
if(queue.isEmpty()){
    System.out.println("Antrian Kosong");
}
```

6. Mencetak panjang queue

```
// method size() mengembalikan panjang queue/antrian
int size = queue.size();
System.out.println("Panjang Antrian " + size);
```

7. Mencetak antrian pertama/head

```
// method peek() mengembalikan nilai dari head
```

```
int first = queue.peek();
System.out.println(“Antrian Pertama : ” + first);
```

8. Menghapus antrian

```
//poll() dan remove() sama-sama menghapus head di queue,
bedanya kalau queue kosong poll() akan melempar NoSuchElementException,
remove() akan mereturn null
queue.poll();
queue.remove();

// Menghapus semua antrian
queue.removeAll(queue);
```

Contoh Program

```
import java.util.*;

public class BelajarQueue{
	public static void main (String[] args){
		Queue<String> antrianBank = new LinkedList<>();

		// Menambah data baru ke list queue (Enqueue)
		antrianBank.add("Adi");
		antrianBank.add("Budi");
		antrianBank.add("Ani");
		antrianBank.add("Didi");

		// Mencetak list queue
		System.out.println("List Antrian Bank : " + antrianBank);

		// Menghapus elemen dari queue (dequeue)
		String nama = antrianBank.remove();
		System.out.println("Menghapus " + nama + "| Antrian baru " + antrianBank);
		// Mencetak Panjang queue
		int banyakAntrian = antrianBank.size();
		System.out.println("Panjang Antrian : " + banyakAntrian);
	}
}
```

Tambahan, sekedar menambah wawasan saja.
Berikut ini adalah implementasi queue menggunakan array:

```
// Java program for array implementation of queue

// A class to represent a queue
class Queue
{
	int front, rear, size;
	int capacity;
	int array[];

	public Queue(int capacity) {
		this.capacity = capacity;
		front = this.size = 0;
		rear = capacity - 1;
		array = new int[this.capacity];

	}

	// Queue is full when size becomes equal to
	// the capacity
	boolean isFull(Queue queue)
	{ return (queue.size == queue.capacity);
	}

	// Queue is empty when size is 0
	boolean isEmpty(Queue queue)
	{ return (queue.size == 0); }

	// Method to add an item to the queue.
	// It changes rear and size
	void enqueue( int item)
	{
		if (isFull(this))
			return;
		this.rear = (this.rear + 1)%this.capacity;
		this.array[this.rear] = item;
		this.size = this.size + 1;
		System.out.println(item+ " enqueued to queue");
	}

	// Method to remove an item from queue.
	// It changes front and size
	int dequeue()
	{
		if (isEmpty(this))
			return Integer.MIN_VALUE;

		int item = this.array[this.front];
		this.front = (this.front + 1)%this.capacity;
		this.size = this.size - 1;
		return item;
	}

	// Method to get front of queue
	int front()
	{
		if (isEmpty(this))
			return Integer.MIN_VALUE;

		return this.array[this.front];
	}

	// Method to get rear of queue
```

```
    int rear()
    {
        if (isEmpty(this))
            return Integer.MIN_VALUE;

        return this.array[this.rear];
    }
}


// Driver class
public class Test
{
    public static void main(String[] args)
    {
        Queue queue = new Queue(1000);

        queue.enqueue(10);
        queue.enqueue(20);
        queue.enqueue(30);
        queue.enqueue(40);

        System.out.println(queue.dequeue() +
                    " dequeued from queue\n");

        System.out.println("Front item is " +
                                queue.front());

        System.out.println("Rear item is " +
                                    queue.rear());
    }
}

// This code is contributed by Gaurav Miglani
```