ava Queue Interface

In this tutorial, we will learn about the Java Queue interface and its methods.

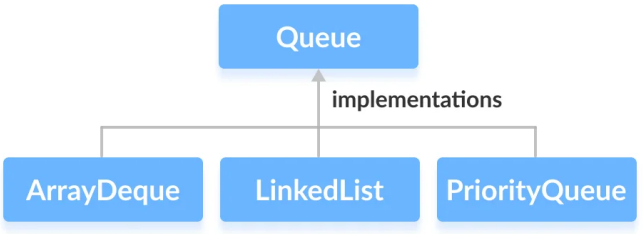
The `Queue` interface of the Java collections framework provides the functionality of the queue data structure. It extends the `Collection` interface.

Classes that Implement Queue

Since the `Queue` is an interface, we cannot provide the direct implementation of it.

In order to use the functionalities of `Queue`, we need to use classes that implement it:

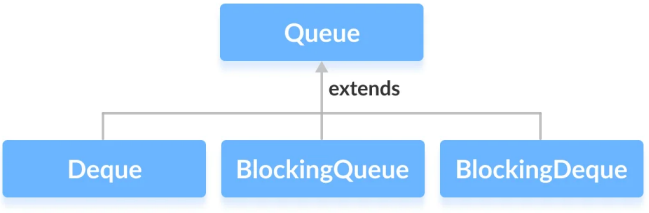
- ArrayDeque
- LinkedList
- PriorityQueue



Interfaces that extend Queue

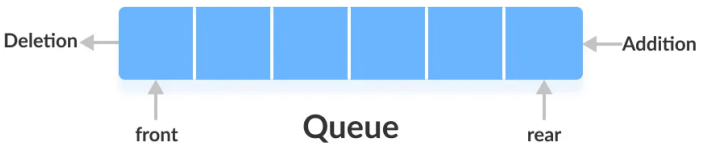
The `Queue` interface is also extended by various subinterfaces:

- `Deque`
- `BlockingQueue`
- `BlockingDeque`



Working of Queue Data Structure

In queues, elements are stored and accessed in **First In, First Out** manner. That is, elements are **added from the behind** and **removed from the front**.



How to use Queue?

In Java, we must import `java.util.Queue` package in order to use `Queue`.

```
// LinkedList implementation of Queue
Queue<String> animal1 = new LinkedList<>();

// Array implementation of Queue
Queue<String> animal2 = new ArrayDeque<>();

// Priority Queue implementation of Queue
Queue<String> animal 3 = new PriorityQueue<>();
```

Here, we have created objects `animal1`, `animal2` and `animal3` of classes `LinkedList`, `ArrayDeque` and `PriorityQueue` respectiv
These objects can use the functionalities of the `Queue` interface.

Methods of Queue

The `Queue` interface includes all the methods of the `Collection` interface. It is because `Collection` is the super interface of `Que`

Some of the commonly used methods of the `Queue` interface are:

- **add()** - Inserts the specified element into the queue. If the task is successful, `add()` returns `true`, if not it throws an exceptio
- **offer()** - Inserts the specified element into the queue. If the task is successful, `offer()` returns `true`, if not it returns `false`.
- **element()** - Returns the head of the queue. Throws an exception if the queue is empty.
- **peek()** - Returns the head of the queue. Returns `null` if the queue is empty.
- **remove()** - Returns and removes the head of the queue. Throws an exception if the queue is empty.
- **poll()** - Returns and removes the head of the queue. Returns `null` if the queue is empty.

1. Implementing the LinkedList Class

```
import java.util.Queue;
import java.util.LinkedList;

class Main {

    public static void main(String[] args) {
        // Creating Queue using the LinkedList class
        Queue<Integer> numbers = new LinkedList<>();

        // offer elements to the Queue
        numbers.offer(1);
        numbers.offer(2);
        numbers.offer(3);
        System.out.println("Queue: " + numbers);

        // Access elements of the Queue
        int accessedNumber = numbers.peek();
        System.out.println("Accessed Element: " + accessedNumber);

        // Remove elements from the Queue
        int removedNumber = numbers.poll();
        System.out.println("Removed Element: " + removedNumber);

        System.out.println("Updated Queue: " + numbers);
    }
}
```

Run Code

Output

```
Queue: [1, 2, 3]
Accessed Element: 1
Removed Element: 1
Updated Queue: [2, 3]
```

To learn more, visit Java LinkedList.

2. Implementing the PriorityQueue Class

```java
import java.util.Queue;
import java.util.PriorityQueue;

class Main {

    public static void main(String[] args) {
        // Creating Queue using the PriorityQueue class
        Queue<Integer> numbers = new PriorityQueue<>();

        // offer elements to the Queue
        numbers.offer(5);
        numbers.offer(1);
        numbers.offer(2);
        System.out.println("Queue: " + numbers);

        // Access elements of the Queue
        int accessedNumber = numbers.peek();
        System.out.println("Accessed Element: " + accessedNumber);

        // Remove elements from the Queue
        int removedNumber = numbers.poll();
        System.out.println("Removed Element: " + removedNumber);

        System.out.println("Updated Queue: " + numbers);
    }
}
```

Run Code

Output

```
Queue: [1, 5, 2]
Accessed Element: 1
Removed Element: 1
Updated Queue: [2, 5]
```