Deque interface in Java with Example

Read Discuss Practice

Deque interface present in java.util package is a subtype of the queue interface. The Deque is related to the double-ended queue that supports the addition or removal of elements from either end of the data structure. It can either be used as a queue(first-in-first-out/FIFO) or as a stack(last-in-first-out/LIFO). Deque is the acronym for double-ended queue.

The Deque (double-ended queue) interface in Java is a subinterface of the Queue interface and extends it to provide a double-ended queue, which is a queue that allows elements to be added and removed from both ends. The Deque interface is part of the Java Collections Framework and is used to provide a generic and flexible data structure that can be used to implement a variety of algorithms and data structures.

Here's an example of how you might use a Deque in Java:

Java

```java
import java.util.ArrayDeque;
import java.util.Deque;

public class Example {
  public static void main(String[] args) {
    Deque<Integer> deque = new ArrayDeque<>();
    deque.addFirst(1);
    deque.addLast(2);
    int first = deque.removeFirst();
    int last = deque.removeLast();
    System.out.println("First: " + first + ", Last: " + last);
  }
}
```

Output

```
First: 1, Last: 2
```

Advantages of using Deque:

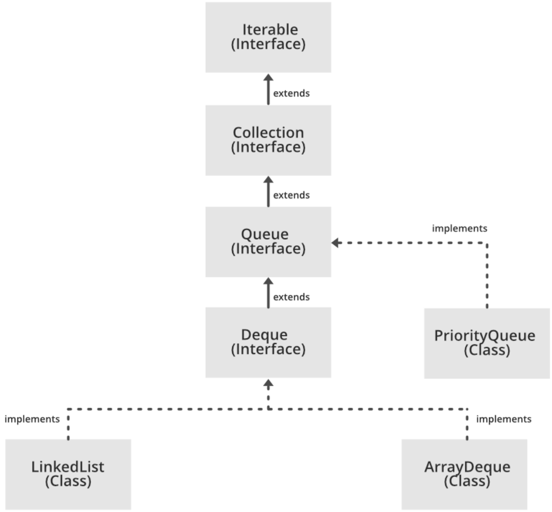
1. Double-Ended: The main advantage of the Deque interface is that it provides a double-ended queue, which allows elements to be added and removed from both ends of the queue. This makes it a good choice for scenarios where you need to insert or remove elements at both the front and end of the queue.
2. Flexibility: The Deque interface provides a number of methods for adding, removing, and retrieving elements from both ends of the queue, giving you a great deal of flexibility in how you use it.
3. Blocking Operations: The Deque interface provides blocking methods, such as takeFirst and takeLast, that allow you to wait for elements to become available or for space to become available in the queue. This makes it a good choice for concurrent and multithreaded applications.

Disadvantages of using Deque:

1. Performance: The performance of a Deque can be slower than other data structures, such as a linked list or an array, because it provides more functionality.
2. Implementation Dependent: The behavior of a Deque can depend on the implementation you use. For example, some implementations may provide thread-safe operations, while others may not. It's important to choose an appropriate implementation and understand its behavior before using a Deque.

Reference Book:

"Java Generics and Collections" by Maurice Naftalin and Philip Wadler is a comprehensive guide to the Java Collections Framework and generics, including the Deque interface. This book covers the basics of the Collections Framework, explains how to use collections and generics in practice, and provides tips and best practices for writing correct and efficient code. If you're looking to learn more about the Deque interface and the Java Collections Framework in general, this book is an excellent resource.



Syntax: The deque interface is declared as:

> public interface Deque extends Queue

Creating Deque Objects Since Deque is an interface, objects cannot be created of the type deque. We always need a class that extends this list in order to create an object. And also, after the introduction of Generics in Java 1.5, it is possible to restrict the type of object that can be stored in the Deque. This type-safe queue can be defined as:

> // Obj is the type of the object to be stored in Deque Deque<Obj> deque = new ArrayDeque<Obj> ();

Example: Deque

Java

```java
// Java program to demonstrate the working
// of a Deque in Java

import java.util.*;

public class DequeExample {
    public static void main(String[] args)
    {
        Deque<String> deque
            = new LinkedList<String>();

        // We can add elements to the queue
        // in various ways

        // Add at the last
        deque.add("Element 1 (Tail)");

        // Add at the first
        deque.addFirst("Element 2 (Head)");

        // Add at the last
        deque.addLast("Element 3 (Tail)");

        // Add at the first
        deque.push("Element 4 (Head)");

        // Add at the last
        deque.offer("Element 5 (Tail)");

        // Add at the first
        deque.offerFirst("Element 6 (Head)");

        System.out.println(deque + "\n");

        // We can remove the first element
        // or the last element.
        deque.removeFirst();
        deque.removeLast();
        System.out.println("Deque after removing "
                        + "first and last: "
                        + deque);
    }
}
```

Output

```
[Element 6 (Head), Element 4 (Head), Element 2 (Head), Element 1 (Tail), Element 3 (Tail),
Element 5 (Tail)]

Deque after removing first and last: [Element 4 (Head), Element 2 (Head), Element 1 (Tail),
Element 3 (Tail)]
```

Operations using the Deque Interface and the ArrayDeque class

Let's see how to perform a few frequently used operations on the deque using the ArrayDeque class.

1. Adding Elements: In order to add an element in a deque, we can use the add() method. The difference between a queue and a deque is that in deque, the addition is possible from any direction. Therefore, there are other two methods available named addFirst() and addLast() which are used to add the elements at either end.

Java

```
// Java program to demonstrate the
// addition of elements in deque
```

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

        // add() method to insert
        dq.add("For");
        dq.addFirst("Geeks");
        dq.addLast("Geeks");

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

Output

```
[Geeks, For, Geeks]
```

2. Removing Elements: In order to remove an element from a deque, there are various methods available. Since we can also remove from both ends, the deque interface provides us with *removeFirst()*, *removeLast()* methods. Apart from that, this interface also provides us with the poll(), pop(), pollFirst(), pollLast() methods where pop() is used to remove and return the head of the deque. However, poll() is used because this offers the same functionality as pop() and doesn’t return an exception when the deque is empty.

Java

```java
// Java program to demonstrate the
// removal of elements in deque

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

        // add() method to insert
        dq.add("For");
        dq.addFirst("Geeks");
        dq.addLast("Geeks");

        System.out.println(dq);

        System.out.println(dq.pop());
        System.out.println(dq.poll());

        System.out.println(dq.pollFirst());

        System.out.println(dq.pollLast());
    }
}
```

Output

```
[Geeks, For, Geeks]
Geeks
For
Geeks
null
```

3. Iterating through the Deque: Since a deque can be iterated from both directions, the iterator method of the deque interface provides us two ways to iterate. One from the first and the other from the back.

Java

```
// Java program to demonstrate the
// iteration of elements in deque

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

        // add() method to insert
        dq.add("For");
        dq.addFirst("Geeks");
        dq.addLast("Geeks");
        dq.add("is so good");

        for (Iterator itr = dq.iterator();
            itr.hasNext();) {
            System.out.print(itr.next() + " ");
        }

        System.out.println();

        for (Iterator itr = dq.descendingIterator();
            itr.hasNext();) {
            System.out.print(itr.next() + " ");
        }
    }
}
```

Output

```
Geeks For Geeks is so good
is so good Geeks For Geeks
```

The class which implements the Deque interface is ArrayDeque.

ArrayDeque: ArrayDeque class which is implemented in the collection framework provides us with a way to apply resizable-array. This is a special kind of array that grows and allows users to add or remove an element from both sides of the queue. Array deques have no capacity restrictions and they grow as necessary to support usage. They are not thread-safe which means that in the absence of external synchronization, ArrayDeque does not support concurrent access by multiple threads. ArrayDeque class is likely to be faster than Stack when used as a stack. ArrayDeque class is likely to be faster than LinkedList when used as a queue. Let’s see how to create a queue object using this class.

Java

```java
// Java program to demonstrate the
// creation of deque object using the
// ArrayDeque class in 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);
    }
}
```

Output

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

Methods of Deque Interface

The following are the methods present in the deque interface:

Method	Description
add(element)	This method is used to add an element at the tail of the queue. If the Deque is capacity restricted and no space is left for insertion, it returns an IllegalStateException. The function returns true on successful insertion.
addFirst(element)	This method is used to add an element at the head of the queue. If the Deque is capacity restricted and no space is left for insertion, it returns an IllegalStateException. The function returns true on successful insertion.
addLast(element)	This method is used to add an element at the tail of the queue. If the Deque is capacity restricted and no space is left for insertion, it returns an IllegalStateException. The function returns true on successful insertion.
contains()	This method is used to check whether the queue contains the given object or not.
descendingIterator()	This method returns an iterator for the deque. The elements will be returned in order from last(tail) to first(head).

Method	Description
element()	This method is used to retrieve, but not remove, the head of the queue represented by this deque.
getFirst()	This method is used to retrieve, but not remove, the first element of this deque.
getLast()	This method is used to retrieve, but not remove, the last element of this deque.
iterator()	This method returns an iterator for the deque. The elements will be returned in order from first (head) to last (tail).
offer(element)	This method is used to add an element at the tail of the queue. This method is preferable to add() method since this method does not throws an exception when the capacity of the container is full since it returns false.
offerFirst(element)	This method is used to add an element at the head of the queue. This method is preferable to addFirst() method since this method does not throws an exception when the capacity of the container is full since it returns false.
offerLast(element)	This method is used to add an element at the tail of the queue. This method is preferable to add() method since this method does not throws an exception when the capacity of the container is full since it returns false.
peek()	This method is used to retrieve the element at the head of the deque but doesn’t remove the element from the deque. This method returns null if the deque is empty.
peekFirst()	This method is used to retrieve the element at the head of the deque but doesn’t remove the element from the deque. This method returns null if the deque is empty.
peekLast()	This method is used to retrieve the element at the tail of the deque but doesn’t remove the element from the deque. This method

Method	Description
	returns null if the deque is empty.
poll()	This method is used to retrieve and remove the element at the head of the deque. This method returns null if the deque is empty.
pollFirst()	This method is used to retrieve and remove the element at the head of the deque. This method returns null if the deque is empty.
pollLast()	This method is used to retrieve and remove the element at the tail of the deque. This method returns null if the deque is empty.
pop()	This method is used to remove an element from the head and return it.
push(element)	This method is used to add an element at the head of the queue.
removeFirst()	This method is used to remove an element from the head of the queue.
removeLast()	This method is used to remove an element from the tail of the queue.
size()	This method is used to find and return the size of the deque.

Last Updated : 10 Apr, 2023

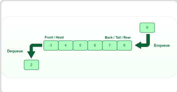
100

Similar Reads

Deque::front() and deque::back() in C++ STL

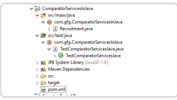
deque::clear() and deque::erase() in C++ STL

deque::operator= and deque::operator[] in C++ STL



Difference between Queue and Deque (Queue vs. Deque)

deque::at() and deque::swap() in C++ STL



Why to Use Comparator Interface Rather than Comparable Interface in Java?

Java 8 | DoubleToIntFunction Interface in Java with Example

Map.Entry interface in Java with example

Java 8 | ObjLongConsumer Interface with Example

Java 8 | ObjIntConsumer Interface with Example

Related Tutorials

Java AWT Tutorial

Spring MVC Tutorial

Spring Tutorial

Spring Boot Tutorial

Java 8 Features - Complete Tutorial

Previous

Next

Article Contributed By :

GeeksforGeeks

Vote for difficulty

Current difficulty : Easy

Easy Normal Medium Hard Expert

Improved By : Anamitra Musib, Akash Gajjar, bipin_kumar, KaashyapMSK, tarunkanade, nandinigujral, rathoadavinash

Article Tags : deque, Java-Collections, java-deque, Java

Practice Tags : Java, Java-Collections

Improve Article Report Issue

GeeksforGeeks

A-143, 9th Floor, Sovereign Corporate
Tower, Sector-136, Noida, Uttar Pradesh -
201305

feedback@geeksforgeeks.org

Company

About Us

Legal

Careers

In Media

Contact Us

Advertise with us

Explore

Job-A-Thon For Freshers

Job-A-Thon For Experienced

GfG Weekly Contest

Offline Classes (Delhi/NCR)

DSA in JAVA/C++

Master System Design

Master CP

Languages

Python

Java

C++

PHP

GoLang

SQL

R Language

Android Tutorial

DSA Concepts

Data Structures

Arrays

Strings

Linked List

Algorithms

Searching

Sorting

Mathematical

Dynamic Programming

DSA Roadmaps

DSA for Beginners

Basic DSA Coding Problems

Complete Roadmap To Learn DSA

DSA for FrontEnd Developers

DSA with JavaScript

Top 100 DSA Interview Problems

Web Development

HTML

CSS

JavaScript

Bootstrap

ReactJS

AngularJS

NodeJS

Computer Science

GATE CS Notes

Operating Systems

Computer Network

Database Management System

Software Engineering

Digital Logic Design

Engineering Maths

Python

Python Programming Examples

Django Tutorial

Python Projects

Python Tkinter

OpenCV Python Tutorial

Python Interview Question

Data Science & ML

Data Science With Python

Data Science For Beginner

Machine Learning Tutorial

Maths For Machine Learning

Pandas Tutorial

NumPy Tutorial

NLP Tutorial

Deep Learning Tutorial

DevOps

Git

AWS

Docker

Kubernetes

Azure

GCP

Competitive Programming

Top DSA for CP

Top 50 Tree Problems

Top 50 Graph Problems

Top 50 Array Problems

Top 50 String Problems

Top 50 DP Problems

Top 15 Websites for CP

System Design

What is System Design

Monolithic and Distributed SD

Scalability in SD

Databases in SD

High Level Design or HLD

Low Level Design or LLD

Top SD Interview Questions

Interview Corner

Company Wise Preparation

Preparation for SDE

Experienced Interviews

Internship Interviews

Competitive Programming

Aptitude Preparation

Commerce

Accountancy

Business Studies

Economics

Management

Income Tax

Finance

SSC/ BANKING

SSC CGL Syllabus

SBI PO Syllabus

SBI Clerk Syllabus

IBPS PO Syllabus

IBPS Clerk Syllabus

Aptitude Questions

SSC CGL Practice Papers

GfG School

CBSE Notes for Class 8

CBSE Notes for Class 9

CBSE Notes for Class 10

CBSE Notes for Class 11

CBSE Notes for Class 12

English Grammar

UPSC

Polity Notes

Geography Notes

History Notes

Science and Technology Notes

Economics Notes

Important Topics in Ethics

UPSC Previous Year Papers

Write & Earn

Write an Article

Improve an Article

Pick Topics to Write

Write Interview Experience

Internships