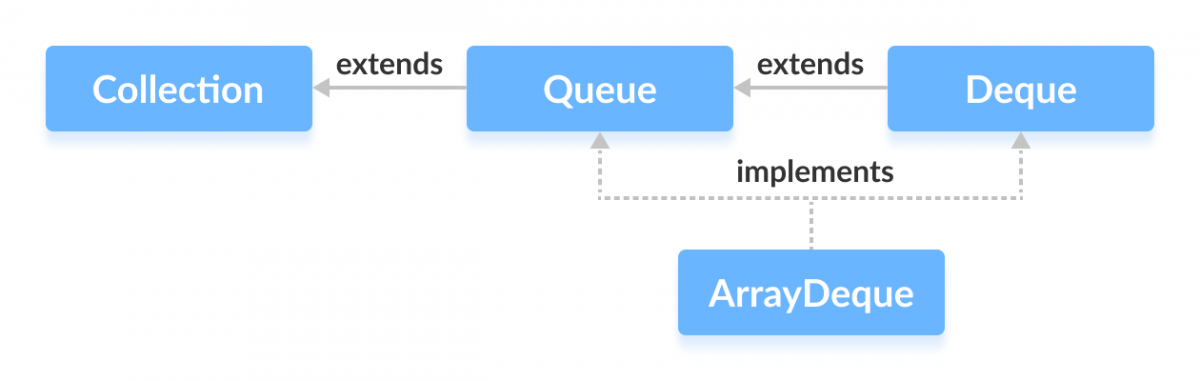
**Java ArrayDeque**

In Java, we can use the ArrayDeque class to implement queue and deque data structures using arrays.

## Interfaces implemented by ArrayDeque

The ArrayDeque class implements these two interfaces:

* Java Queue Interface
* Java Deque Interface



## Creating ArrayDeque

In order to create an array deque, we must import the java.util.ArrayDeque package.

Here is how we can create an array deque in Java:

ArrayDeque<Type> animal = new ArrayDeque<>();

Here, Type indicates the type of the array deque. For example,

// Creating String type ArrayDeque

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

// Creating Integer type ArrayDeque

ArrayDeque<Integer> age = new ArrayDeque<>();

## Methods of ArrayDeque

The ArrayDeque class provides implementations for all the methods present in Queue and Deque interface.

### Insert Elements to Deque

**1. Add elements using add(), addFirst() and addLast()**

* add() - inserts the specified element at the end of the array deque
* addFirst() - inserts the specified element at the beginning of the array deque
* addLast() - inserts the specified at the end of the array deque (equivalent to add())

**Note:** If the array deque is full, all these methods add(), addFirst() and addLast() throws IllegalStateException.

For example,

import java.util.ArrayDeque;

class Main {

public static void main(String[] args) {

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

// Using add()

animals.add("Dog");

// Using addFirst()

animals.addFirst("Cat");

// Using addLast()

animals.addLast("Horse");

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

}

}

**Output**

ArrayDeque: [Cat, Dog, Horse]

**2. Insert elements using offer(), offerFirst() and offerLast()**

* offer() - inserts the specified element at the end of the array deque
* offerFirst() - inserts the specified element at the beginning of the array deque
* offerLast() - inserts the specified element at the end of the array deque

**Note:** offer(), offerFirst() and offerLast() returns true if the element is successfully inserted; if the array deque is full, these methods return false.

For example,

import java.util.ArrayDeque;

class Main {

public static void main(String[] args) {

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

// Using offer()

animals.offer("Dog");

// Using offerFirst()

animals.offerFirst("Cat");

// Using offerLast()

animals.offerLast("Horse");

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

}

}

**Output**

ArrayDeque: [Cat, Dog, Horse]

**Note:** If the array deque is full

* the add() method will throw an exception
* the offer() method returns false

### Access ArrayDeque Elements

**1. Access elements using getFirst() and getLast()**

* getFirst() - returns the first element of the array deque
* getLast() - returns the last element of the array deque

**Note:** If the array deque is empty, getFirst() and getLast() throws NoSuchElementException.

For example.

import java.util.ArrayDeque;

class Main {

public static void main(String[] args) {

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

animals.add("Dog");

animals.add("Cat");

animals.add("Horse");

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

// Get the first element

String firstElement = animals.getFirst();

System.out.println("First Element: " + firstElement);

// Get the last element

String lastElement = animals.getLast();

System.out.println("Last Element: " + lastElement);

}

}

**Output**

ArrayDeque: [Dog, Cat, Horse]

First Element: Dog

Last Element: Horse

**2. Access elements using peek(), peekFirst() and peekLast() method**

* peek() - returns the first element of the array deque
* peekFirst() - returns the first element of the array deque (equivalent to peek())
* peekLast() - returns the last element of the array deque

For example,

import java.util.ArrayDeque;

class Main {

public static void main(String[] args) {

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

animals.add("Dog");

animals.add("Cat");

animals.add("Horse");

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

// Using peek()

String element = animals.peek();

System.out.println("Head Element: " + element);

// Using peekFirst()

String firstElement = animals.peekFirst();

System.out.println("First Element: " + firstElement);

// Using peekLast

String lastElement = animals.peekLast();

System.out.println("Last Element: " + lastElement);

}

}

**Output**

ArrayDeque: [Dog, Cat, Horse]

Head Element: Dog

First Element: Dog

Last Element: Horse

**Note:** If the array deque is empty, peek(), peekFirst() and getLast() throws NoSuchElementException.

### Remove ArrayDeque Elements

**1. Remove elements using the remove(), removeFirst(), removeLast() method**

* remove() - returns and removes an element from the first element of the array deque
* remove(element) - returns and removes the specified element from the head of the array deque
* removeFirst() - returns and removes the first element from the array deque (equivalent to remove())
* removeLast() - returns and removes the last element from the array deque

**Note:** If the array deque is empty, remove(), removeFirst() and removeLast() method throws an exception. Also, remove(element) throws an exception if the element is not found.

For example,

import java.util.ArrayDeque;

class Main {

public static void main(String[] args) {

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

animals.add("Dog");

animals.add("Cat");

animals.add("Cow");

animals.add("Horse");

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

// Using remove()

String element = animals.remove();

System.out.println("Removed Element: " + element);

System.out.println("New ArrayDeque: " + animals);

// Using removeFirst()

String firstElement = animals.removeFirst();

System.out.println("Removed First Element: " + firstElement);

// Using removeLast()

String lastElement = animals.removeLast();

System.out.println("Removed Last Element: " + lastElement);

}

}

**Output**

ArrayDeque: [Dog, Cat, Cow, Horse]

Removed Element: Dog

New ArrayDeque: [Cat, Cow, Horse]

Removed First Element: Cat

Removed Last Element: Horse

**2. Remove elements using the poll(), pollFirst() and pollLast() method**

* poll() - returns and removes the first element of the array deque
* pollFirst() - returns and removes the first element of the array deque (equivalent to poll())
* pollLast() - returns and removes the last element of the array deque

**Note:** If the array deque is empty, poll(), pollFirst() and pollLast() returns null if the element is not found.

For example,

import java.util.ArrayDeque;

class Main {

public static void main(String[] args) {

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

animals.add("Dog");

animals.add("Cat");

animals.add("Cow");

animals.add("Horse");

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

// Using poll()

String element = animals.poll();

System.out.println("Removed Element: " + element);

System.out.println("New ArrayDeque: " + animals);

// Using pollFirst()

String firstElement = animals.pollFirst();

System.out.println("Removed First Element: " + firstElement);

// Using pollLast()

String lastElement = animals.pollLast();

System.out.println("Removed Last Element: " + lastElement);

}

}

**Output**

ArrayDeque: [Dog, Cat, Cow, Horse]

Removed Element: Dog

New ArrayDeque: [Cat, Cow, Horse]

Removed First Element: Cat

Removed Last Element: Horse

**3. Remove Element: using the clear() method**

To remove all the elements from the array deque, we use the clear() method. For example,

import java.util.ArrayDeque;

class Main {

public static void main(String[] args) {

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

animals.add("Dog");

animals.add("Cat");

animals.add("Horse");

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

// Using clear()

animals.clear();

System.out.println("New ArrayDeque: " + animals);

}

}

**Output**

ArrayDeque: [Dog, Cat, Horse]

New ArrayDeque: []

### Iterating the ArrayDeque

* iterator() - returns an iterator that can be used to iterate over the array deque
* descendingIterator() - returns an iterator that can be used to iterate over the array deque in reverse order

In order to use these methods, we must import the java.util.Iterator package. For example,

import java.util.ArrayDeque;

import java.util.Iterator;

class Main {

public static void main(String[] args) {

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

animals.add("Dog");

animals.add("Cat");

animals.add("Horse");

System.out.print("ArrayDeque: ");

// Using iterator()

Iterator<String> iterate = animals.iterator();

while(iterate.hasNext()) {

System.out.print(iterate.next());

System.out.print(", ");

}

System.out.print("\nArrayDeque in reverse order: ");

// Using descendingIterator()

Iterator<String> desIterate = animals.descendingIterator();

while(desIterate.hasNext()) {

System.out.print(desIterate.next());

System.out.print(", ");

}

}

}

**Output**

ArrayDeque: [Dog, Cat, Horse]

ArrayDeque in reverse order: [Horse, Cat, Dog]

## Other Methods

|  |  |
| --- | --- |
| Methods | Descriptions |
| element() | Returns an element from the head of the array deque. |
| contains(element) | Searches the array deque for the specified element. If the element is found, it returns true, if not it returns false. |
| size() | Returns the length of the array deque. |
| toArray() | Converts array deque to array and returns it. |
| clone() | Creates a copy of the array deque and returns it. |

## ArrayDeque as a Stack

To implement a **LIFO (Last-In-First-Out)** stacks in Java, it is recommended to use a deque over the Stack class. The ArrayDeque class is likely to be faster than the Stack class.

ArrayDeque provides the following methods that can be used for implementing a stack.

* push() - adds an element to the top of the stack
* peek() - returns an element from the top of the stack
* pop() - returns and removes an element from the top of the stack

For example,

import java.util.ArrayDeque;

class Main {

public static void main(String[] args) {

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

// Add elements to stack

stack.push("Dog");

stack.push("Cat");

stack.push("Horse");

System.out.println("Stack: " + stack);

// Access element from top of stack

String element = stack.peek();

System.out.println("Accessed Element: " + element);

// Remove elements from top of stack

String remElement = stack.pop();

System.out.println("Removed element: " + remElement);

}

}

**Output**

Stack: [Horse, Cat, Dog]

Accessed Element: Horse

Removed Element: Horse

## ArrayDeque Vs. LinkedList Class

Both ArrayDeque and Java LinkedList implements the Deque interface. However, there exist some differences between them.

* LinkedList supports null elements, whereas ArrayDeque doesn't.
* Each node in a linked list includes links to other nodes. That's why LinkedList requires more storage than ArrayDeque.
* If you are implementing the queue or the deque data structure, an ArrayDeque is likely to faster than a LinkedList.