ArrayDeque in Java

Difficulty Level : Easy Last Updated : 24 Jan, 2022

The ArrayDeque in Java provides a way to apply resizable-array in addition to the implementation of the Deque interface. It is also known as ***Array Double Ended Queue*** or ***Array Deck***. This is a special kind of array that grows and allows users to add or remove an element from both sides of the queue.

Few ***important features*** of ArrayDeque are as follows:

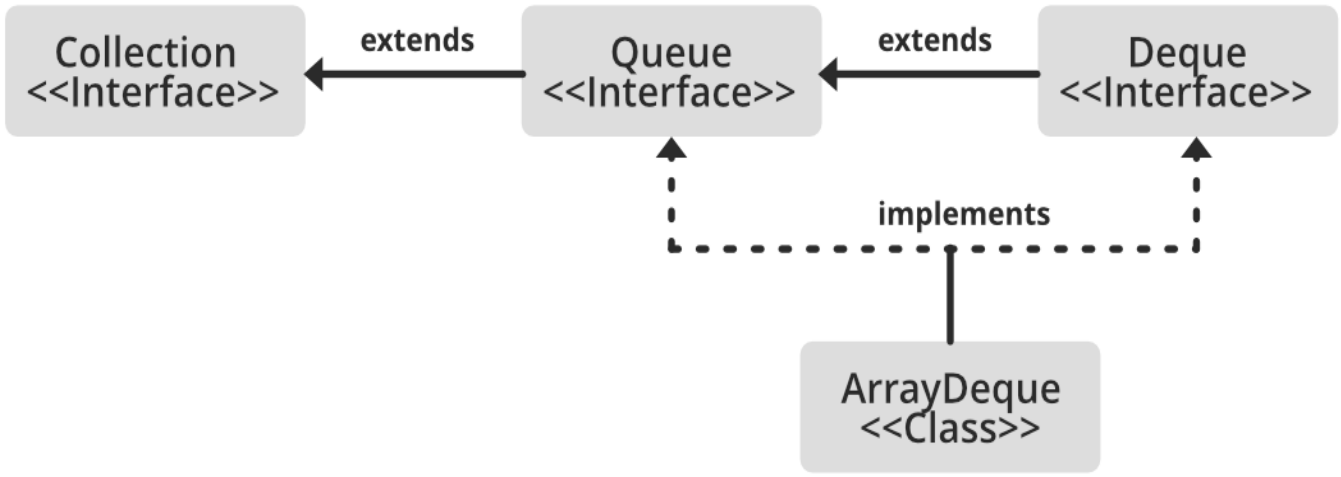
- Array deques have no capacity restrictions and they grow as necessary to support usage.
- They are not thread-safe which means that in the absence of external synchronization, ArrayDeque does not support concurrent access by multiple threads.
- Null elements are prohibited in the ArrayDeque.
- ArrayDeque 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.

Interfaces implemented by ArrayDeque:

The ArrayDeque class implements these two interfaces:

- **Queue Interface:** It is an Interface that is a FirstIn – FirstOut Data Structure where the elements are added from the back.
- **Deque Interface:** It is a Doubly Ended Queue in which you can insert the elements from both sides. It is an interface that implements the Queue.

ArrayDeque implements both Queue and Deque. It is dynamically resizable from both sides. All implemented interfaces of ArrayDeque in the hierarchy are **Serializable**, **Cloneable**, **Iterable<E>**, **Collection<E>**, Deque<E>, Queue<E>



Syntax: Declaration

public class ArrayDeque<E>

extends *AbstractCollection<E>*

implements *Deque<E>, Cloneable, Serializable*

*Here, **E** refers to the element which can refer to any class, such as Integer or String class.*

Now we are done with syntax now let us come up with constructors been defined for it prior before implementing to grasp it better and perceiving the output better.

- **ArrayDeque():** This constructor is used to create an empty ArrayDeque and by default holds an initial capacity to hold 16 elements.

```
ArrayDeque<E> dq = new ArrayDeque<E>();
```

- **ArrayDeque(Collection<? extends E> c):** This constructor is used to create an ArrayDeque containing all the elements the same as that of the specified collection.

```
ArrayDeque<E> dq = new ArrayDeque<E>(Collection col);
```

- **ArrayDeque(int numofElements):** This constructor is used to create an empty ArrayDeque and holds the capacity to contain a specified number of elements.

```
ArrayDeque<E> dq = new ArrayDeque<E>(int numofElements);
```

Methods in ArrayDeque are as follows:

Note: *Here,* ***Element*** *is the type of elements stored by ArrayDeque.*

METHOD	DESCRIPTION
add(Element e)	The method inserts a particular element at the end of the deque.
addAll(Collection<? extends E> c)	Adds all of the elements in the specified collection at the end of this deque, as if by calling addLast(E) on each one, in the order that they are returned by the collection's iterator.
addFirst(Element e)	The method inserts particular element at the start of the deque.
addLast(Element e)	The method inserts a particular element at the end of the deque. It is similar to the add() method
clear()	The method removes all deque elements.
clone()	The method copies the deque.
contains(Obj)	The method checks whether a deque contains the element or not
element()	The method returns element at the head of the deque
forEach(Consumer<? super E> action)	Performs the given action for each element of the Iterable until all elements have been processed or the action throws an exception.
getFirst()	The method returns first element of the deque

METHOD	DESCRIPTION
getLast()	The method returns last element of the deque
isEmpty()	The method checks whether the deque is empty or not.
iterator()	Returns an iterator over the elements in this deque.
offer(Element e)	The method inserts element at the end of deque.
offerFirst(Element e)	The method inserts element at the front of deque.
offerLast(Element e)	The method inserts element at the end of the deque.
peek()	The method returns head element without removing it.
poll()	The method returns head element and also removes it
pop()	The method pops out an element for stack represented by deque
push(Element e)	The method pushes an element onto stack represented by deque
remove()	The method returns head element and also removes it
remove(Object o)	Removes a single instance of the specified element from this deque.
removeAll(Collection<? > c)	Removes all of this collection’s elements that are also contained in the specified collection (optional operation).
removeFirst()	The method returns the first element and also removes it
removeFirstOccurrence (Object o)	Removes the first occurrence of the specified element in this deque (when traversing the deque from head to tail).
removeIf(Predicate<? super Element> filter)	Removes all of the elements of this collection that satisfy the given predicate.
removeLast()	The method returns the last element and also removes it

METHOD	DESCRIPTION
removeLastOccurrence (Object o)	Removes the last occurrence of the specified element in this deque (when traversing the deque from head to tail).
retainAll(Collection<?> c)	Retains only the elements in this collection that are contained in the specified collection (optional operation).
size()	Returns the number of elements in this deque.
spliterator()	Creates a late-binding and fail-fast Spliterator over the elements in this deque.
toArray()	Returns an array containing all of the elements in this deque in proper sequence (from first to the last element).
toArray(T[] a)	Returns an array containing all of the elements in this deque in proper sequence (from first to the last element); the runtime type of the returned array is that of the specified array.

Methods inherited from class java.util.AbstractCollection

Method	Action Performed
containsAll(Collection c)	Returns true if this collection contains all of the elements in the specified collection.
toString()	Returns a string representation of this collection.

Methods inherited from interface java.util.Collection

Method	Action Performed
containsAll(Collection c)	Returns true if this collection contains all of the elements in the specified collection.
equals()	Compares the specified object with this collection for equality.
hashcode()	Returns the hash code value for this collection.

Method	Action Performed
parallelStream()	Returns a possibly parallel Stream with this collection as its source.
stream()	Returns a sequential Stream with this collection as its source.
toArray (IntFunction<T[]> generator)	Returns an array containing all of the elements in this collection, using the provided generator function to allocate the returned array.

Methods declared in interface java.util.Deque

Method	Action Performed
descendingIterator()	Returns an iterator over the elements in this deque in reverse sequential order.
peekFirst()	Retrieves, but does not remove, the first element of this deque, or returns null if this deque is empty.
peekLast()	Retrieves, but does not remove, the last element of this deque, or returns null if this deque is empty.
pollFirst()	Retrieves and removes the first element of this deque, or returns null if this deque is empty.
pollLast()	Retrieves and removes the last element of this deque, or returns null if this deque is empty.

Example

```
// Java program to Implement ArrayDeque in Java
//

// Importing utility classes
import java.util.*;
```

```
// ArrayDequeDemo
public class GGFG {
    public static void main(String[] args)
    {
        // Creating and initializing deque
        // Declaring object of integer type
        Deque<Integer> de_que = new ArrayDeque<Integer>(10);

        // Operations 1
        // add() method

        // Adding custom elements
        // using add() method to insert
        de_que.add(10);
        de_que.add(20);
        de_que.add(30);
        de_que.add(40);
        de_que.add(50);

        // Iterating using for each loop
        for (Integer element : de_que) {
            // Print the corresponding element
            System.out.println("Element : " + element);
        }

        // Operation 2
        // clear() method
        System.out.println("Using clear() ");

        // Clearing all elements using clear() method
        de_que.clear();

        // Operations 3
        // addFirst() method

        // Inserting at the start
        de_que.addFirst(564);
        de_que.addFirst(291);

        // Operation 4
        // addLast() method
        // Inserting at end
        de_que.addLast(24);
        de_que.addLast(14);

        // Display message
        System.out.println(
            "Above elements are removed now");

        // Iterators

        // Display message
        System.out.println(
            "Elements of deque using Iterator :");
```

```
        for (Iterator itr = de_que.iterator();
             itr.hasNext();) {
            System.out.println(itr.next());
        }

        // descendingIterator()
        // To reverse the deque order
        System.out.println(
            "Elements of deque in reverse order :");

        for (Iterator dItr = de_que.descendingIterator();
             dItr.hasNext();) {
            System.out.println(dItr.next());
        }

        // Operation 5
        // element() method : to get Head element
        System.out.println(
            "\nHead Element using element(): "
            + de_que.element());

        // Operation 6
        // getFirst() method : to get Head element
        System.out.println("Head Element using getFirst(): "
                           + de_que.getFirst());

        // Operation 7
        // getLast() method : to get last element
        System.out.println("Last Element using getLast(): "
                           + de_que.getLast());

        // Operation 8
        // toArray() method :
        Object[] arr = de_que.toArray();
        System.out.println("\nArray Size : " + arr.length);

        System.out.print("Array elements : ");

        for (int i = 0; i < arr.length; i++)
            System.out.print(" " + arr[i]);

        // Operation 9
        // peek() method : to get head
        System.out.println("\nHead element : "
                           + de_que.peek());

        // Operation 10
        // poll() method : to get head
        System.out.println("Head element poll : "
                           + de_que.poll());

        // Operation 11
        // push() method
        de_que.push(265);
        de_que.push(984);
        de_que.push(2365);
```

```
            // Operation 12
            // remove() method : to get head
            System.out.println("Head element remove : "
                               + de_que.remove());

            System.out.println("The final array is: " + de_que);
        }
    }
```

Output

```
Element : 10
Element : 20
Element : 30
Element : 40
Element : 50
Using clear()
Above elements are removed now
Elements of deque using Iterator :
291
564
24
14
Elements of deque in reverse order :
14
24
564
291

Head Element using element(): 291
Head Element using getFirst(): 291
Last Element using getLast(): 14

Array Size : 4
Array elements :  291 564 24 14
Head element : 291
Head element poll : 291
Head element remove : 2365
The final array is: [984, 265, 564, 24, 14]
```

If there is some lag in clarity in this example, if so then we are proposing various operations on the ArrayDeque class Let's see how to perform a few frequently used operations on the ArrayDeque to get a better understanding of the operations that we have used above to illustrate Array Deque as a whole.

- Adding operation
- Accessing operation
- Removing operations
- Iterating through the Deque

Let us go through each of the operations by implementing alongside by providing clean java program as follows:

Operation 1: Adding Elements

In order to add an element to the ArrayDeque, we can use the methods add(), addFirst(), addLast(), offer(), offerFirst(), offerLast() methods.

- add()
- addFirst()
- addLast()
- offer()
- offerFirst()
- offerLast()

Example

```
// Java program to Illustrate Addition of elements
// in ArrayDeque

// Importing required classes
import java.io.*;
import java.util.*;

// Main class
// AddingElementsToArrayDeque
public class GFG {

    // Main driver method
    public static void main(String[] args)
    {
```

```
        // Initializing a deque
        // since deque is an interface
        // it is assigned the
        // ArrayDeque class
        Deque<String> dq = new ArrayDeque<String>();

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

        // offer() method to insert
        dq.offer("For");
        dq.offerFirst("Welcome");
        dq.offerLast("Geeks");

        // Printing Elements of ArrayDeque to the console
        System.out.println("ArrayDeque : " + dq);
    }
}
```

Output:

```
ArrayDeque : [Welcome, To, The, Geeks, For, Geeks]
```

Operation 2: Accessing the Elements

After adding the elements, if we wish to access the elements, we can use inbuilt methods like getFirst(), getLast(), etc.

- getFirst()
- getLast()
- peek()
- peekFirst()
- peekLast()

Example

```
// Java program to Access Elements of ArrayDeque

// Importing required classes
import java.io.*;
```

```
import java.util.*;

// Main class
// AccessingElementsOfArrayDeque
public class GFG {

    // Main driver method
    public static void main(String args[])
    {
        // Creating an empty ArrayDeque
        ArrayDeque<String> de_que
            = new ArrayDeque<String>();

        // Using add() method to add elements into the Deque
        // Custom input elements
        de_que.add("Welcome");
        de_que.add("To");
        de_que.add("Geeks");
        de_que.add("4");
        de_que.add("Geeks");

        // Displaying the ArrayDeque
        System.out.println("ArrayDeque: " + de_que);

        // Displaying the First element
        System.out.println("The first element is: "
                           + de_que.getFirst());

        // Displaying the Last element
        System.out.println("The last element is: "
                           + de_que.getLast());
    }
}
```

Output:

```
ArrayDeque: [Welcome, To, Geeks, 4, Geeks]
The first element is: Welcome
The last element is: Geeks
```

Operation 3. Removing Elements

In order to remove an element from a deque, there are various methods available. Since we can also remove from both the 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. These sets of operations are as listed below as follows:

- remove()
- removeFirst()
- removeLast()
- poll()
- pollFirst()
- pollLast()
- pop()

Example

```
// Java program to Illustrate Removal Elements in Deque

// Importing all utility classes
import java.util.*;

// RemoveElementsOfArrayDeque
public class GFG {

    // Main driver method
    public static void main(String[] args)
    {
        // Initializing a deque
        Deque<String> dq = new ArrayDeque<String>();

        // add() method to insert
        dq.add("One");

        // addFirst inserts at the front
        dq.addFirst("Two");

        // addLast inserts at the back
        dq.addLast("Three");

        // print elements to the console
        System.out.println("ArrayDeque : " + dq);

        // remove element as a stack from top/front
        System.out.println(dq.pop());

        // remove element as a queue from front
        System.out.println(dq.poll());
```

```
        // remove element from front
        System.out.println(dq.pollFirst());

        // remove element from back
        System.out.println(dq.pollLast());
    }
}
```

Output

```
ArrayDeque : [Two, One, Three]
Two
One
Three
null
```

Operation 4: 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. These sets of operations are listed below as follows:

- remove()
- iterator()
- descendingIterator()

Example

```
// Java program to Illustrate Iteration of Elements
// in Deque

// Importing all utility classes
import java.util.*;

// Main class
// IterateArrayDeque
public class GFG {

    // Main driver method
    public static void main(String[] args)
```

```
    {
        // Declaring and initializing an deque
        Deque<String> dq = new ArrayDeque<String>();

        // Adding elements at the back
        // using add() method
        dq.add("For");

        // Adding element at the front
        // using addFirst() method
        dq.addFirst("Geeks");

        // add element at the last
        // using addLast() method
        dq.addLast("Geeks");
        dq.add("is so good");

        // Iterate using Iterator interface
        // from the front of the queue
        for (Iterator itr = dq.iterator(); itr.hasNext();) {

            // Print the elements
            System.out.print(itr.next() + " ");
        }

        // New line
        System.out.println();

        // Iterate in reverse sequence in a queue
        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
```

Related Articles:

- Java.util.ArrayDeque Class in Java | Set 1
- Java.util.ArrayDeque Class in Java | Set 2
- ArrayList vs LinkedList in Java