Java Queue Interface

The interface Queue is available in the java.util package and does extend the Collection interface. It is used to keep the elements that are processed in the First In First Out (FIFO) manner. It is an ordered list of objects, where insertion of elements occurs at the end of the list, and removal of elements occur at the beginning of the list.

Being an interface, the queue requires, for the declaration, a concrete class, and the most common classes are the LinkedList and PriorityQueue in Java. Implementations done by these classes are not thread safe. If it is required to have a thread safe implementation, PriorityBlockingQueue is an available option.

Queue Interface Declaration

```
public interface Queue<E> extends Collection<E>
```

Methods of Java Queue Interface

Method	Description
boolean add(object)	It is used to insert the specified element into this queue and return true upon success.
boolean offer(object)	It is used to insert the specified element into this queue.
Object remove()	It is used to retrieves and removes the head of this queue.
Object poll()	It is used to retrieves and removes the head of this queue, or returns null if this queue is empty.
Object element()	It is used to retrieves, but does not remove, the head of this queue.
Object peek()	It is used to retrieves, but does not remove, the head of this queue, or returns null if this queue is empty.

Features of a Queue

The following are some important features of a queue.

- As discussed earlier, FIFO concept is used for insertion and deletion of elements from a queue.
- The Java Queue provides support for all of the methods of the Collection interface including deletion, insertion, etc.
- PriorityQueue, ArrayBlockingQueue and LinkedList are the implementations that are used most frequently.

- The NullPointerException is raised, if any null operation is done on the BlockingQueues.
- Those Queues that are present in the *util* package are known as Unbounded Queues.
- Those Queues that are present in the *util.concurrent* package are known as bounded Queues.
- All Queues barring the Deques facilitates removal and insertion at the head and tail of the queue; respectively. In fact, deques support element insertion and removal at both ends.

PriorityQueue Class

PriorityQueue is also class that is defined in the collection framework that gives us a way for processing the objects on the basis of priority. It is already described that the insertion and deletion of objects follows FIFO pattern in the Java queue. However, sometimes the elements of the queue are needed to be processed according to the priority, that's where a PriorityQueue comes into action.

PriorityQueue Class Declaration

Let's see the declaration for java.util.PriorityQueue class.

```java
public class PriorityQueue<E> extends AbstractQueue<E> implements Serializable
```

Java PriorityQueue Example

FileName: TestCollection12.java

```java
import java.util.*;
class TestCollection12{
public static void main(String args[]){
PriorityQueue<String> queue=new PriorityQueue<String>();
queue.add("Amit");
queue.add("Vijay");
queue.add("Karan");
queue.add("Jai");
queue.add("Rahul");
System.out.println("head:"+queue.element());
System.out.println("head:"+queue.peek());
System.out.println("iterating the queue elements:");
Iterator itr=queue.iterator();
while(itr.hasNext()){
System.out.println(itr.next());
}
queue.remove();
queue.poll();
System.out.println("after removing two elements:");
Iterator<String> itr2=queue.iterator();
while(itr2.hasNext()){
System.out.println(itr2.next());
}
}
}
```

Test it Now

Output:

```
head:Amit
       head:Amit
       iterating the queue elements:
       Amit
       Jai
       Karan
       Vijay
       Rahul
       after removing two elements:
       Karan
       Rahul
       Vijay
```

Java PriorityQueue Example: Book

Let's see a PriorityQueue example where we are adding books to queue and printing all the books. The elements in PriorityQueue must be of Comparable type. String and Wrapper classes are Comparable by default. To add user-defined objects in PriorityQueue, you need to implement Comparable interface.

FileName: LinkedListExample.java

```
import java.util.*;
class Book implements Comparable<Book>{
int id;
String name,author,publisher;
int quantity;
public Book(int id, String name, String author, String publisher, int quantity) {

    this.id = id;
    this.name = name;
    this.author = author;
    this.publisher = publisher;
    this.quantity = quantity;
}
```

```java
public int compareTo(Book b) {
    if(id>b.id){
        return 1;
    }else if(id<b.id){
        return -1;
    }else{
    return 0;
    }
}
}
public class LinkedListExample {
public static void main(String[] args) {
    Queue<Book> queue=new PriorityQueue<Book>();
    //Creating Books
    Book b1=new Book(121,"Let us C","Yashwant Kanetkar","BPB",8);
    Book b2=new Book(233,"Operating System","Galvin","Wiley",6);
    Book b3=new Book(101,"Data Communications & Networking","Forouzan","Mc Graw Hill",4);
    //Adding Books to the queue
    queue.add(b1);
    queue.add(b2);
    queue.add(b3);
    System.out.println("Traversing the queue elements:");
    //Traversing queue elements
    for(Book b:queue){
    System.out.println(b.id+" "+b.name+" "+b.author+" "+b.publisher+" "+b.quantity);
    }
    queue.remove();
    System.out.println("After removing one book record:");
    for(Book b:queue){
        System.out.println(b.id+" "+b.name+" "+b.author+" "+b.publisher+" "+b.quantity);
        }
}
}
```

Output:

```
Traversing the queue elements:
101 Data Communications & Networking Forouzan Mc Graw Hill 4
233 Operating System Galvin Wiley 6
121 Let us C Yashwant Kanetkar BPB 8
After removing one book record:
121 Let us C Yashwant Kanetkar BPB 8
233 Operating System Galvin Wiley 6
```

Java Deque Interface

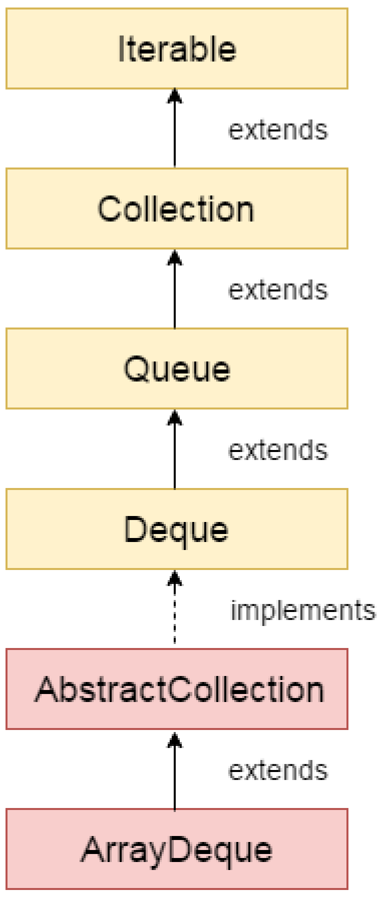
The interface called Deque is present in java.util package. It is the subtype of the interface queue. The Deque supports the addition as well as the removal of elements from both ends of the data structure. Therefore, a deque can be used as a stack or a queue. We know that the stack supports the Last In First Out (LIFO) operation, and the operation First In First Out is supported by a queue. As a deque supports both, either of the mentioned operations can be performed on it. Deque is an acronym for **"double ended queue".**

Deque Interface declaration

```
public interface Deque<E> extends Queue<E>
```

Methods of Java Deque Interface

Method	Description
boolean add(object)	It is used to insert the specified element into this deque and return true upon success.
boolean offer(object)	It is used to insert the specified element into this deque.
Object remove()	It is used to retrieve and removes the head of this deque.
Object poll()	It is used to retrieve and removes the head of this deque, or returns null if this deque is empty.
Object element()	It is used to retrieve, but does not remove, the head of this deque.
Object peek()	It is used to retrieve, but does not remove, the head of this deque, or returns null if this deque is empty.
Object peekFirst()	The method returns the head element of the deque. The method does not remove any element from the deque. Null is returned by this method, when the deque is empty.
Object peekLast()	The method returns the last element of the deque. The method does not remove any element from the deque. Null is returned by this method, when the deque is empty.
Boolean offerFirst(e)	Inserts the element e at the front of the queue. If the insertion is successful, true is returned; otherwise, false.
Object offerLast(e)	Inserts the element e at the tail of the queue. If the insertion is successful, true is returned; otherwise, false.



ArrayDeque class

We know that it is not possible to create an object of an interface in Java. Therefore, for instantiation, we need a class that implements the Deque interface, and that class is ArrayDeque. It grows and shrinks as per usage. It also inherits the AbstractCollection class.

The important points about ArrayDeque class are:

- Unlike Queue, we can add or remove elements from both sides.
- Null elements are not allowed in the ArrayDeque.
- ArrayDeque is not thread safe, in the absence of external synchronization.
- ArrayDeque has no capacity restrictions.
- ArrayDeque is faster than LinkedList and Stack.

ArrayDeque Hierarchy

The hierarchy of ArrayDeque class is given in the figure displayed at the right side of the page.

ArrayDeque class declaration

Let's see the declaration for java.util.ArrayDeque class.

```
public class ArrayDeque<E> extends AbstractCollection<E> implements Deque<E>, Cloneable, Serializable
```

Java ArrayDeque Example

FileName: ArrayDequeExample.java

```java
import java.util.*;
public class ArrayDequeExample {
  public static void main(String[] args) {
  //Creating Deque and adding elements
  Deque<String> deque = new ArrayDeque<String>();
  deque.add("Ravi");
  deque.add("Vijay");
  deque.add("Ajay");
  //Traversing elements
  for (String str : deque) {
  System.out.println(str);
  }
  }
}
```

Output:

```
Ravi
Vijay
Ajay
```

Java ArrayDeque Example: offerFirst() and pollLast()

FileName: DequeExample.java

```java
import java.util.*;
public class DequeExample {
public static void main(String[] args) {
   Deque<String> deque=new ArrayDeque<String>();
   deque.offer("arvind");
   deque.offer("vimal");
   deque.add("mukul");
   deque.offerFirst("jai");
   System.out.println("After offerFirst Traversal...");
   for(String s:deque){
       System.out.println(s);
   }
   //deque.poll();
   //deque.pollFirst();//it is same as poll()
   deque.pollLast();
   System.out.println("After pollLast() Traversal...");
```

```java
    for(String s:deque){
        System.out.println(s);
    }
}
}
```

Output:

```
After offerFirst Traversal...
jai
arvind
vimal
mukul
After pollLast() Traversal...
jai
arvind
vimal
```

Java ArrayDeque Example: Book

FileName: ArrayDequeExample.java

```java
import java.util.*;
class Book {
int id;
String name,author,publisher;
int quantity;
public Book(int id, String name, String author, String publisher, int quantity) {

    this.id = id;
    this.name = name;
    this.author = author;
    this.publisher = publisher;
    this.quantity = quantity;
}
}
public class ArrayDequeExample {
public static void main(String[] args) {
    Deque<Book> set=new ArrayDeque<Book>();
    //Creating Books
    Book b1=new Book(101,"Let us C","Yashwant Kanetkar","BPB",8);
    Book b2=new Book(102,"Data Communications & Networking","Forouzan",
"Mc Graw Hill",4);
    Book b3=new Book(103,"Operating System","Galvin","Wiley",6);
    //Adding Books to Deque
    set.add(b1);
    set.add(b2);
    set.add(b3);
    //Traversing ArrayDeque
    for(Book b:set){
    System.out.println(b.id+" "+b.name+" "+b.author+" "+b.publisher+" "+b.quantity);
    }
}
}
```

Output:

```
101 Let us C Yashwant Kanetkar BPB 8
102 Data Communications & Networking Forouzan Mc Graw Hill 4
103 Operating System Galvin Wiley 6
```