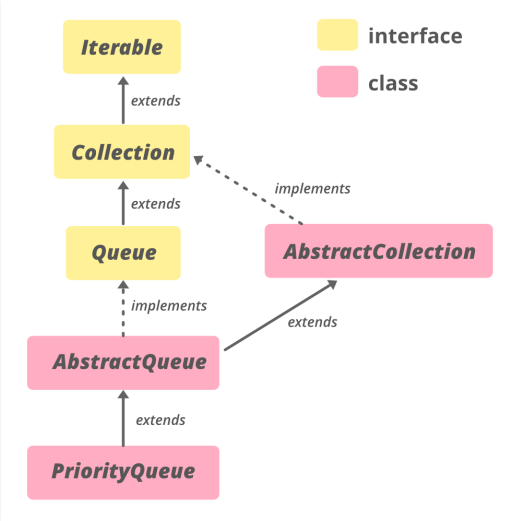
AbstractQueue in Java with Examples

Difficulty Level : Basic Last Updated : 05 Nov, 2020

The **AbstractQueue** class in Java is a part of the Java Collection Framework and implements the ***Collection*** *interface* and the *AbstractCollection class*. It provides skeletal implementations of some **Queue** operations. The implementations in this class are appropriate when the base implementation does not allow null elements. Methods add, remove, and element are based on offer, poll, and peek, respectively, but throw exceptions instead of indicating failure via false or null returns.

Class Hierarchy:

```
java.lang.Object
 ↳ java.util.AbstractCollection<E>
    ↳ Class AbstractQueue<E>
```



This class implements **Iterable<E>**, **Collection<E>**, Queue<E> interfaces and extends AbstractCollection

Declaration:

public abstract class AbstractQueue<E> extends AbstractCollection<E> implements Queue<E>

E – Type of element maintained by the Collection Framework class or interface.

Constructors in Java AbstractQueue

Since AbstractQueue is an abstract class, it's implementation is provided by its sub-classes. Below shows the list of classes that can provide the implementation. To create it, we need to it from **java.uti.AbstractQueue**.

protected AbstractQueue(): The default constructor, but being abstract, it doesn't allow to create an AbstractQueue object. The implementation should be provided by one of its subclasses like ArrayBlockingQueue, ConcurrentLinkedQueue, DelayQueue, LinkedBlockingDeque, LinkedBlockingQueue, LinkedTransferQueue, PriorityBlockingQueue, PriorityQueue, **SynchronousQueue**.

```
AbstractQueue<E> objName = new ArrayBlockingQueue<E>();
```

Below is a sample program to illustrate AbstractQueue in Java:

```
// Java code to illustrate AbstractQueue

import java.util.*;
import java.util.concurrent.LinkedBlockingQueue;

public class AbstractQueueExample {

    public static void main(String[] argv) throws Exception
    {
        // Creating object of AbstractQueue<Integer>
        AbstractQueue<Integer> AQ = new LinkedBlockingQueue<Integer>();

        // Adding elements to the Queue
        AQ.add(10);
        AQ.add(20);
        AQ.add(30);
        AQ.add(40);
        AQ.add(50);

        // print the queue contents to the console
        System.out.println("AbstractQueue contains: " + AQ);
    }
}
```

Output:

```
AbstractQueue contains: [10, 20, 30, 40, 50]
```

Basic Operations

1. Adding Elements

To add elements into the AbstractQueue, it provides two methods. The add(E e) method inserts the specified element into this queue if it is possible to do so immediately without violating capacity

restrictions. It returns true upon success and throws an **IllegalStateException** if no space is currently available. The addAll(E e) method adds all the elements in the specified collection to this queue.

```
// Java program to illustrate the
// adding elements to the AbstractQueue

import java.util.*;
import java.util.concurrent.LinkedBlockingQueue;

public class AddingElementsExample {

    public static void main(String[] argv) throws Exception
    {
        // Since AbstractQueue is an abstract class
        // create object using LinkedBlockingQueue
        AbstractQueue<Integer> AQ1 = new LinkedBlockingQueue<Integer>();

        // Populating AQ
        AQ1.add(10);
        AQ1.add(20);
        AQ1.add(30);
        AQ1.add(40);
        AQ1.add(50);

        // print AQ
        System.out.println("AbstractQueue contains : "
                           + AQ1);

        // Since AbstractQueue is an abstract class
        // create object using LinkedBlockingQueue
        AbstractQueue<Integer> AQ2 = new LinkedBlockingQueue<Integer>();

        // print AQ2 initially
        System.out.println("AbstractQueue2 initially contains : " + AQ2);

        // adds elements of AQ1 in AQ2
        AQ2.addAll(AQ1);

        System.out.println( "AbstractQueue1 after addition contains : " + AQ2);
    }
}
```

Output

```
AbstractQueue contains : [10, 20, 30, 40, 50]
AbstractQueue2 initially contains : []
AbstractQueue1 after addition contains : [10, 20, 30, 40, 50]
```

2. Remove the Elements

To remove the elements from AbstractQueue, it provides remove() and clear() methods.

- The remove() method returns and removes the head of this queue.
- The clear() method removes all the elements from this queue. The queue will be empty after this call returns.

```
// Java program to illustrate the
// removal of elements from AbstractQueue

import java.util.*;
import java.util.concurrent.LinkedBlockingQueue;

public class RemovingElementsExample {
    public static void main(String[] argv) throws Exception
    {
        // Since AbstractQueue is an abstract class
        // create object using LinkedBlockingQueue
        AbstractQueue<Integer> AQ1 = new LinkedBlockingQueue<Integer>();

        // Add elements using add method
        AQ1.add(10);
        AQ1.add(20);
        AQ1.add(30);
        AQ1.add(40);
        AQ1.add(50);

        // print the queue contents to the console
        System.out.println("AbstractQueue1 contains : " + AQ1);

        // Retrieves the head
        int head = AQ1.remove();

        // print the head element to the console
        System.out.println("head : " + head);

        // print the modified queue
        System.out.println("AbstractQueue1 after removal of head : " + AQ1);

        // remove all the elements
        AQ1.clear();

        // print the modified queue
        System.out.println("AbstractQueue1 : " + AQ1);
    }
}
```

Output

```
AbstractQueue1 contains : [10, 20, 30, 40, 50]
head : 10
AbstractQueue1 after removal of head : [20, 30, 40, 50]
AbstractQueue1 : []
```

3. Accessing the Elements

The element() method of AbstractQueue retrieves but does not remove, the head of this queue.

```
// Java program to illustrate the
// accessing element from AbstractQueue

import java.util.*;
import java.util.concurrent.LinkedBlockingQueue;

public class AccessingElementExample {

    public static void main(String[] argv) throws Exception
    {
        // Since AbstractQueue is an abstract class
        // create object using LinkedBlockingQueue
        AbstractQueue<Integer> AQ1 = new LinkedBlockingQueue<Integer>();

        // Populating AQ1 using add method
        AQ1.add(10);
        AQ1.add(20);
        AQ1.add(30);
        AQ1.add(40);
        AQ1.add(50);

        // print AQ to the console
        System.out.println("AbstractQueue1 contains : " + AQ1);

        // access the head element
        System.out.println("head : " + AQ1.element());
    }
}
```

Output

```
AbstractQueue1 contains : [10, 20, 30, 40, 50]
head : 10
```

Methods of AbstractQueue

METHOD	DESCRIPTION
add(E e)	Inserts the specified element into this queue if it is possible to do so immediately without violating capacity restrictions, returning true upon success, and throwing an IllegalStateException if no space is currently available.
addAll (Collection<? extends E> c)	Adds all the elements in the specified collection to this queue.
clear()	Removes all the elements from this queue.
element()	Retrieves, but does not remove, the head of this queue.
remove()	Retrieves and removes the head of this queue.

Methods declared in class java.util.AbstractCollection

METHOD	DESCRIPTION
contains(Object o)	Returns true if this collection contains the specified element.
containsAll (Collection<?> c)	Returns true if this collection contains all of the elements in the specified collection.
isEmpty()	Returns true if this collection contains no elements.
iterator()	Returns an iterator over the elements contained in this collection.
remove(Object o)	Removes a single instance of the specified element from this collection, if it is present (optional operation).
removeAll (Collection<?> c)	Removes all of this collection’s elements that are also contained in the specified collection (optional operation).
retainAll (Collection<?> c)	Retains only the elements in this collection that are contained in the specified collection (optional operation).
toArray()	Returns an array containing all of the elements in this collection.
toArray(T[] a)	Returns an array containing all of the elements in this collection; the runtime type of the returned array is that of the specified array.
toString()	Returns a string representation of this collection.

Methods declared in interface java.util.Collection

METHOD	DESCRIPTION
contains(Object o)	Returns true if this collection contains the specified element.
containsAll (Collection<?> c)	Returns true if this collection contains all the elements in the specified collection.
equals(Object o)	Compares the specified object with this collection for equality.
hashCode()	Returns the hash code value for this collection.
isEmpty()	Returns true if this collection contains no elements.
iterator()	Returns an iterator over the elements in this collection.
parallelStream()	Returns a possibly parallel Stream with this collection as its source.
remove(Object o)	Removes a single instance of the specified element from this collection, if it is present (optional operation).
removeAll (Collection<?> c)	Removes all of this collection's elements that are also contained in the specified collection (optional operation).
removeIf(Predicate<? super E> filter)	Removes all the elements of this collection that satisfy the given predicate.
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 collection.
spliterator()	Creates a Spliterator over the elements in this collection.
stream()	Returns a sequential Stream with this collection as its source.
toArray()	Returns an array containing all the elements in this collection.
toArray (IntFunction<T[]> generator)	Returns an array containing all the elements in this collection, using the provided generator function to allocate the returned array.
toArray(T[] a)	Returns an array containing all the elements in this collection; the runtime type of the returned array is that of the specified array.

Methods declared in interface java.lang.Iterable

METHOD	DESCRIPTION
forEach(Consumer<? super T> action)	Performs the given action for each element of the Iterable until all elements have been processed or the action throws an exception.

Methods declared in interface java.util.Queue

METHOD	DESCRIPTION
offer(E e)	Inserts the specified element into this queue if it is possible to do so immediately without violating capacity restrictions.
peek()	Retrieves, but does not remove, the head of this queue, or returns null if this queue is empty.
poll()	Retrieves and removes the head of this queue, or returns null if this queue is empty.

Reference: https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/AbstractQueue.html