```
/*
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Oracle designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Oracle in the LICENSE file that accompanied this code.
 *
 * This code is distributed in the hope that it will be useful, but WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
 * FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Oracle, 500 Oracle Parkway, Redwood Shores, CA 94065 USA
 * or visit www.oracle.com if you need additional information or have any
 * questions.
 */

/*
 * This file is available under and governed by the GNU General Public
 * License version 2 only, as published by the Free Software Foundation.
 * However, the following notice accompanied the original version of this
 * file:
 *
 * Written by Doug Lea with assistance from members of JCP JSR-166
 * Expert Group and released to the public domain, as explained at
 * http://creativecommons.org/publicdomain/zero/1.0/
 */

package java.util;

/**
 * A collection designed for holding elements prior to processing.
 * Besides basic {@link Collection} operations, queues provide
 * additional insertion, extraction, and inspection operations.
 * Each of these methods exists in two forms: one throws an exception
 * if the operation fails, the other returns a special value (either
 * {@code null} or {@code false}, depending on the operation).  The
 * latter form of the insert operation is designed specifically for
 * use with capacity-restricted {@code Queue} implementations; in most
 * implementations, insert operations cannot fail.
 *
 * <table class="striped">
 * <caption>Summary of Queue methods</caption>
 *  <thead>
 *  <tr>
 *    <td></td>
 *    <th scope="col" style="font-weight:normal; font-style:italic">Throws exception</th>
 *    <th scope="col" style="font-weight:normal; font-style:italic">Returns special value</th>
 *  </tr>
 *  </thead>
 *  <tbody>
 *  <tr>
 *    <th scope="row">Insert</th>
 *    <td>{@link #add(Object) add(e)}</td>
 *    <td>{@link #offer(Object) offer(e)}</td>
 *  </tr>
 *  <tr>
 *    <th scope="row">Remove</th>
 *    <td>{@link #remove() remove()}</td>
 *    <td>{@link #poll() poll()}</td>
 *  </tr>
 *  <tr>
 *    <th scope="row">Examine</th>
 *    <td>{@link #element() element()}</td>
 *    <td>{@link #peek() peek()}</td>
 *  </tr>
 *  </tbody>
 * </table>
 *
 * <p>Queues typically, but do not necessarily, order elements in a
 * FIFO (first-in-first-out) manner.  Among the exceptions are
```

```
 * priority queues, which order elements according to a supplied
 * comparator, or the elements' natural ordering, and LIFO queues (or
 * stacks) which order the elements LIFO (last-in-first-out).
 * Whatever the ordering used, the <em>head</em> of the queue is that
 * element which would be removed by a call to {@link #remove()} or
 * {@link #poll()}.  In a FIFO queue, all new elements are inserted at
 * the <em>tail</em> of the queue. Other kinds of queues may use
 * different placement rules.  Every {@code Queue} implementation
 * must specify its ordering properties.
 *
 * <p>The {@link #offer offer} method inserts an element if possible,
 * otherwise returning {@code false}.  This differs from the {@link
 * java.util.Collection#add Collection.add} method, which can fail to
 * add an element only by throwing an unchecked exception.  The
 * {@code offer} method is designed for use when failure is a normal,
 * rather than exceptional occurrence, for example, in fixed-capacity
 * (or &quot;bounded&quot;) queues.
 *
 * <p>The {@link #remove()} and {@link #poll()} methods remove and
 * return the head of the queue.
 * Exactly which element is removed from the queue is a
 * function of the queue's ordering policy, which differs from
 * implementation to implementation. The {@code remove()} and
 * {@code poll()} methods differ only in their behavior when the
 * queue is empty: the {@code remove()} method throws an exception,
 * while the {@code poll()} method returns {@code null}.
 *
 * <p>The {@link #element()} and {@link #peek()} methods return, but do
 * not remove, the head of the queue.
 *
 * <p>The {@code Queue} interface does not define the <i>blocking queue
 * methods</i>, which are common in concurrent programming.  These methods,
 * which wait for elements to appear or for space to become available, are
 * defined in the {@link java.util.concurrent.BlockingQueue} interface, which
 * extends this interface.
 *
 * <p>{@code Queue} implementations generally do not allow insertion
 * of {@code null} elements, although some implementations, such as
 * {@link LinkedList}, do not prohibit insertion of {@code null}.
 * Even in the implementations that permit it, {@code null} should
 * not be inserted into a {@code Queue}, as {@code null} is also
 * used as a special return value by the {@code poll} method to
 * indicate that the queue contains no elements.
 *
 * <p>{@code Queue} implementations generally do not define
 * element-based versions of methods {@code equals} and
 * {@code hashCode} but instead inherit the identity based versions
 * from class {@code Object}, because element-based equality is not
 * always well-defined for queues with the same elements but different
 * ordering properties.
 *
 * <p>This interface is a member of the
 * <a href="{@docRoot}/java/util/package-summary.html#CollectionsFramework">
 * Java Collections Framework</a>.
 *
 * @since 1.5
 * @author Doug Lea
 * @param <E> the type of elements held in this queue
 */
public interface Queue<E> extends Collection<E> {
    /**
     * Inserts the specified element into this queue if it is possible to do so
     * immediately without violating capacity restrictions, returning
     * {@code true} upon success and throwing an {@code IllegalStateException}
     * if no space is currently available.
     *
     * @param e the element to add
     * @return {@code true} (as specified by {@link Collection#add})
     * @throws IllegalStateException if the element cannot be added at this
     *         time due to capacity restrictions
     * @throws ClassCastException if the class of the specified element
     *         prevents it from being added to this queue
     * @throws NullPointerException if the specified element is null and
     *         this queue does not permit null elements
     * @throws IllegalArgumentException if some property of this element
     *         prevents it from being added to this queue
     */
    boolean add(E e);
```

```
    /**
     * Inserts the specified element into this queue if it is possible to do
     * so immediately without violating capacity restrictions.
     * When using a capacity-restricted queue, this method is generally
     * preferable to {@link #add}, which can fail to insert an element only
     * by throwing an exception.
     *
     * @param e the element to add
     * @return {@code true} if the element was added to this queue, else
     *         {@code false}
     * @throws ClassCastException if the class of the specified element
     *         prevents it from being added to this queue
     * @throws NullPointerException if the specified element is null and
     *         this queue does not permit null elements
     * @throws IllegalArgumentException if some property of this element
     *         prevents it from being added to this queue
     */
    boolean offer(E e);

    /**
     * Retrieves and removes the head of this queue.  This method differs
     * from {@link #poll() poll()} only in that it throws an exception if
     * this queue is empty.
     *
     * @return the head of this queue
     * @throws NoSuchElementException if this queue is empty
     */
    E remove();

    /**
     * Retrieves and removes the head of this queue,
     * or returns {@code null} if this queue is empty.
     *
     * @return the head of this queue, or {@code null} if this queue is empty
     */
    E poll();

    /**
     * Retrieves, but does not remove, the head of this queue.  This method
     * differs from {@link #peek peek} only in that it throws an exception
     * if this queue is empty.
     *
     * @return the head of this queue
     * @throws NoSuchElementException if this queue is empty
     */
    E element();

    /**
     * Retrieves, but does not remove, the head of this queue,
     * or returns {@code null} if this queue is empty.
     *
     * @return the head of this queue, or {@code null} if this queue is empty
     */
    E peek();
}
```