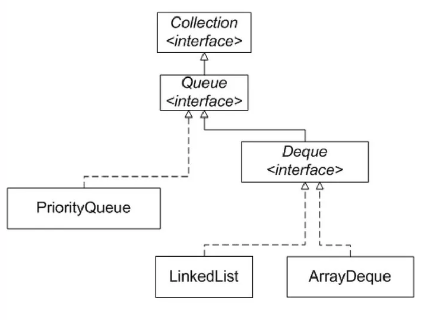
**Queue**

**(ОЧЕРЕДЬ)**



Данная коллекция предназначена для хранения элементов в порядке нужном для их обработки. Обычно представлена в виде FIFO, но может быть организованна как LIFO или как двухнаправленная очередь, стек. Все очереди **не** могут содержать null элементы!

Ограниченные очереди по кол-ву элементов хранятся в папке java.util.concurrent. Не ограниченные очереди (которы е увеличиваются при добавлении элементов) хранятся в java.util.

**Queue (Interface)**

Интерфейс который наследуется прямо от Collection и определяет поведение классов в качестве **однонаправленной** очереди. По технологии FIFO.

Основные методы:

E element() – возвращает, но не удаляет объект из очереди. Если коллекция пустая – исключение **NoSuchElementException**.

E peek() – возвращает, но не удаляет объект из очереди. Если коллекция пустая – возвращает **null**.

boolean offer(E) – добавляет обьект в конец очереди (возвращает **true/false**).

E poll() – возвращает и удаляет элемент из начала очереди, если очередь пуста – возвращает **null**.

E remove() – возвращает и удаляет элемент из начала очереди, если очередь пуста – исключение **NoSuchElementException**.

Методы кидают Exception или возвращают null:



**PriorityQueue**

**(реализация, класс)**

Наследуется от интерфейса Queue. Не является потокобезопасный. Не может содержать null элементы. Может иметь естественный порядок, или порядок установленный с помощью компаратора. Удаление/добавление происходит за О(log(n)). Начальный размер = 11. Реализуем Comparable или Comparator у классов которые будут храниться в коллекции и настраиваем способ сравнения. У этого класса все методы заточены на работу по системе FIFO (pop – берём первый элемент и удаляем, peek – берём первый элемент не удаляя, push – закидываем новый элемент в самый конец), так же работает класс Stack.

**BlockingQueue**

**(интерфейс)**

Наследуется от Queue и определяет поведение синхронизированных очередей, разные потоки к потомкам этого интерфейса могут обращаться одновременно без каких либо проблем. Реализация данного интерфеса – **ArrayBlockingQueue,** ориентированна на работу с несколькими+ потоками (имеются методы take() и put(), которые жду появления или освобождения места в данной реализации очереди, после чего выполняют действия).

**PriorityBlockingQueue**

**(реализация, класс)**

Наследуется от PriorityQueue. Потокобезопасный!

**Deque (Interface)**

Интерфейс наследуется от Queue и определяет поведение двунаправленной очереди, при этом может работать как однонаправленная и как стек (по принципу FIFO/LIFO).

Основные методы:

// ДОБАВЛЕНИЯ:

void addFirst(E) – добавить элемент в **начало** очереди.

void addLast(E) – добавляет элемент в **конец** очереди.

boolean offerFirst(E) – добавляет элемент в **начало** (return true/false).

boolean offerLast(E) – добавляет элемент в **конец** очереди (return true/false).

// ПОЛУЧИТЬ НЕ УДАЛЯЯ:

E getFirst() – возвращает без удаления элемент из начала очереди, если пусто – исключение NoSuchElementException.

E getLast() – возвращает без удаления элемент из конца очереди, если пусто – NoSuchElementException.

E peekFirst() – возвращает без удаления элемент из начала очереди, если очередь пуста – null.

E peekLast() – возвращает без удаления элемент из конца очереди, если пусто – null.

// ПОЛУЧЕНИЕ И УДАЛЕНИЕ:

E pollFirst() - возвращает с удалением элемент из начала очереди, если очередь пуста, возвращает значение null

E pollLast() - возвращает с удалением последний элемент очереди, если очередь пуста, возвращает значение null

E removeFirst – возвращает и удаляет элемент из начала очереди, если пусто – NoSuchElementException.

E removeLast – возвращает и удаляет элемент из конца очереди, если пусто – NoSuchElementException.

boolean removeFirstOccurrence(E) и removeLastOccurrence(E) – удаляет первый/последний элемент найденный в очереди(return true/false).

**//Два этих метода позволяют организовать стек:**

E pop() – возвращает и удаляет элемент из начала очереди, если очередь пуста – исключение NoSuchElementException.

void push(E) – добавляет элемент в самое начало очереди.

**ArrayDeque**

**(реализация, класс)**

Реализация интерфейса Deque. Двунаправленная очередь. Не является потокобезопасной. Не поддерживает элементы **null**. При работе как стека – данный класс будет вероятнее быстрее чем Stack (наследник Vector -> List). И быстрее при работе как очередь, чем LinkedList. Расширяется автоматически.

Конструктор может быть пустой, может быть построен на определённой коллекции, или с определённым размером очереди (изначально размер = 16).

Тем не менее, убирать элемент можно и из середины очереди - так, например, человек в любой момент может махнуть рукой и сказать: "А, я уже три часа тут стою - пойду домой!" (в LinkedList просто удалить элемент по значению или индексу).

Более того, элемент не всегда может влезть в очередь

Например, не сталкивались с тем, что доходит очередь до Вас - и как раз начинается обед?  Или касса закрывается?

**FIFO и LIFO**

FIFO – принцип как с очередью, кто первый в неё встал тот первым и будет обслужен. Так же подойдёт пример с конвейером, что положил на него раньше, тот раньше и будет произведён, выпущен с завода.

LIFO – принцип хорош с грудой тарелок, при её мойке, последнию грязную посуду поставленную в стопку ты берёшь и моешь первой.