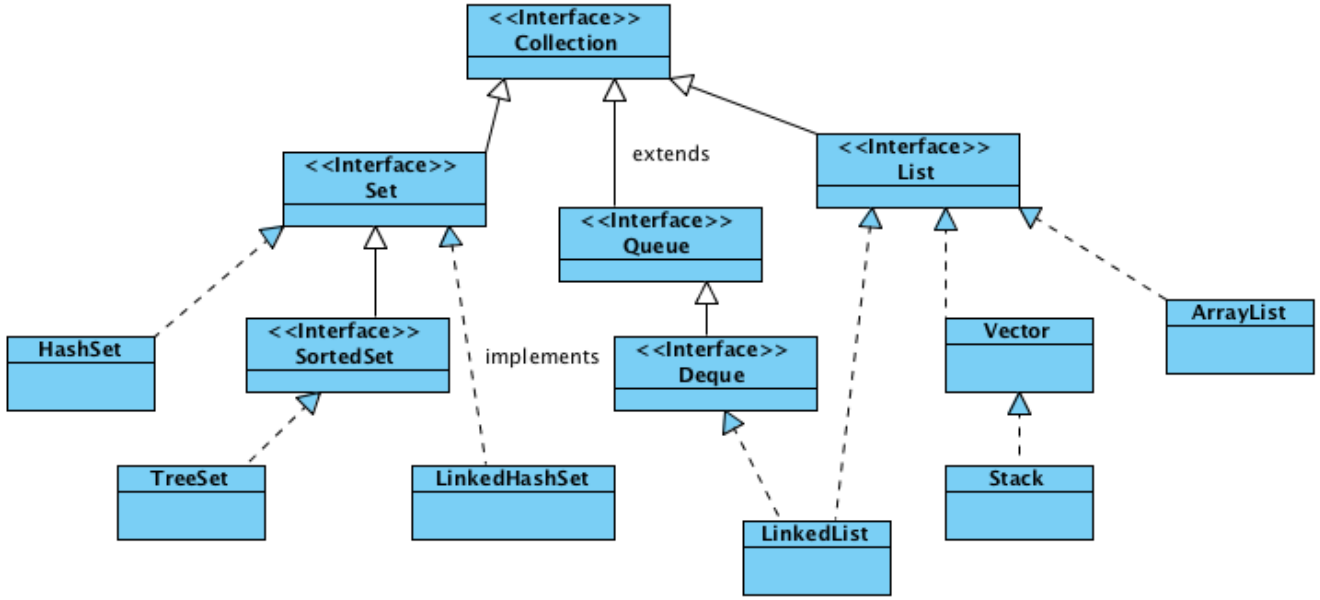
Queue, PriorityQueue, Executors и Queue

Основные вопросы

- Очередь в Java
- PriorityQueue как реализация интерфейса Queue
- Использование очередей в многопоточных приложениях
- Потокобезопасные очереди (пакет util.concurrent)

Очередь – коллекция, предназначенная для хранения элементов в порядке, нужном для их обработки. В дополнение к базовым операциям интерфейса Collection, очередь предоставляет дополнительные операции вставки, получения и контроля. Очереди обычно, но не обязательно, упорядочивают элементы в FIFO (first-in-first-out, "первым вошел - первым вышел") порядке. Не могут хранить значения *null*. Допускают существование одинаковых элементов.



Методы интерфейса Queue

- ***boolean offer(E obj)*** - пытается добавить *obj* в очередь. Возвращает *true*, если *obj* добавлен, и *false* в противном случае.
- ***E element()*** - возвращает элемент из головы очереди. Элемент не удаляется. Если очередь пуста, выбрасывается исключение *NoSuchElementException*.
- ***E peek()*** - возвращает элемент из головы очереди. Возвращает *null*, если очередь пуста. Элемент не удаляется.
- ***E poll()*** - возвращает элемент из головы очереди и удаляет его. Возвращает *null*, если очередь пуста.
- ***E remove()*** - удаляет элемент из головы очереди, возвращая его. Инициирует исключение *NoSuchElementException*, если очередь пуста.

Пример работы с очередью

```
public static void main(String[] args) {
    Queue<Integer> queue = new LinkedList<>();
    queue.offer( e: 1);
    queue.offer( e: 10);
    queue.offer( e: 100);
    System.out.println("Созданная очередь: " + queue);
    System.out.println("Голова очереди (метод element): " + queue.element());
    System.out.println("Голова очереди (метод peek): " + queue.peek());

    queue.remove();
    System.out.println("Очередь после метода remove: " + queue);
    queue.poll();
    System.out.println("Очередь после метода poll: " + queue);
}
```

```
Созданная очередь: [1, 10, 100]
Голова очереди (метод element): 1
Голова очереди (метод peek): 1
Очередь после метода remove: [10, 100]
Очередь после метода poll: [100]
```

Выход из зоны комфорта

```java
public static void main(String[] args) {
    Queue<Integer> queue = new LinkedList<>();
    System.out.println("Созданная очередь: " + queue);

    try {
        System.out.println("Голова очереди (метод element): " + queue.element());
    } catch (NoSuchElementException e) {
        System.out.println("Может, лучше сначала заполнить очередь, а потом вызывать element?");
    }

    System.out.println("Голова очереди (метод peek): " + queue.peek());

    try {
        queue.remove();
    } catch (NoSuchElementException e) {
        System.out.println("Очередь пуста, что с помощью remove удалять-то собрался?");
    }

    System.out.println(queue.poll());
}
```

```
Созданная очередь: []
Может, лучше сначала заполнить очередь, а потом вызывать element?
Голова очереди (метод peek): null
Очередь пуста, что с помощью remove удалять-то собрался?
null
```

PriorityQueue

PriorityQueue – это класс очереди с приоритетами, который является единственной прямой реализацией интерфейса Queue. По умолчанию очередь с приоритетами работает с элементами согласно естественному порядку сортировки используя Comparable. Элементу с наименьшим значением присваивается наибольший приоритет. Если несколько элементов имеют одинаковый наивысший элемент – связь определяется произвольно. Также можно указать специальный порядок присваивания приоритета, используя Comparator.

Пример работы с очередью с приоритетами

```java
public class QueueMain3 {
    public static void main(String[] args) {
        CustomerOrder c1 = new CustomerOrder( orderId: 1,  orderAmount: 27.0,  customerName: "customer1");
        CustomerOrder c2 = new CustomerOrder( orderId: 6,  orderAmount: 50.0,  customerName: "customer2");
        CustomerOrder c3 = new CustomerOrder( orderId: 4,  orderAmount: 43.0,  customerName: "customer3");

        Queue<CustomerOrder> customerOrders = new PriorityQueue<>();
        customerOrders.offer(c1);
        customerOrders.offer(c2);
        customerOrders.offer(c3);
        while (!customerOrders.isEmpty()) {
            System.out.println(customerOrders.poll());
        }
    }
}
```

```java
public class CustomerOrder implements Comparable<CustomerOrder> {
    private int orderId;
    private double orderAmount;
    private String customerName;

    public CustomerOrder(int orderId, double orderAmount, String customerName) {
        this.orderId = orderId;
        this.orderAmount = orderAmount;
        this.customerName = customerName;
    }

    @Override
    public int compareTo(CustomerOrder order) {
        return order.orderId > this.orderId ? 1 : -1;
    }

    @Override
    public String toString() {
        return "Customer name: " + this.customerName + ", order amount: " + this.orderAmount
                + ", order id: " + this.orderId;
    }
}
```

```
Customer name: customer2, order amount: 50.0, order id: 6
Customer name: customer3, order amount: 43.0, order id: 4
Customer name: customer1, order amount: 27.0, order id: 1
```

Пример посложнее

```java
@Override
public void run() {
    int counter = 0;

    try {
        for (int i = 0; i < this.customerList.size(); i++) {
            if (this.queue.size() >= 4) {
                counter++;
                System.out.println("Количество попыток записи в занятую очередь: " + counter);
                i--;
                Thread.sleep( l: 500);

                if (counter == 4) {
                    System.out.println("Слишком много попыток записи в занятую очередь.");
                    CustomerMain.latch.countDown();
                    break;
                }
            } else {
                synchronized (lock) {
                    this.queue.offer(this.customerList.get(i));
                }
                System.out.println("В очередь добавлен: " + this.customerList.get(i).toString());
                Thread.sleep( l: 500);
                counter = 0;
            }
        }
    } catch (InterruptedException e) {
        e.printStackTrace();
    }

    CustomerMain.latch.countDown();
}
```

```java
@Override
public void run() {
    CustomerOrder order;
    while (true) {
        synchronized (CustomerGiver.lock) {
            if (queue.isEmpty()) {
                System.out.println("Очередь пуста. Прекращение работы.");
                CustomerMain.latch.countDown();
                return;
            }

            order = this.queue.poll();
        }

        order.setPrice(order.getOrderAmount() * 10d);
        System.out.println("Обработка очереди: " + order.toString() + ", order price: " + order.getPrice());

        try {
            Thread.sleep( l: 3000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

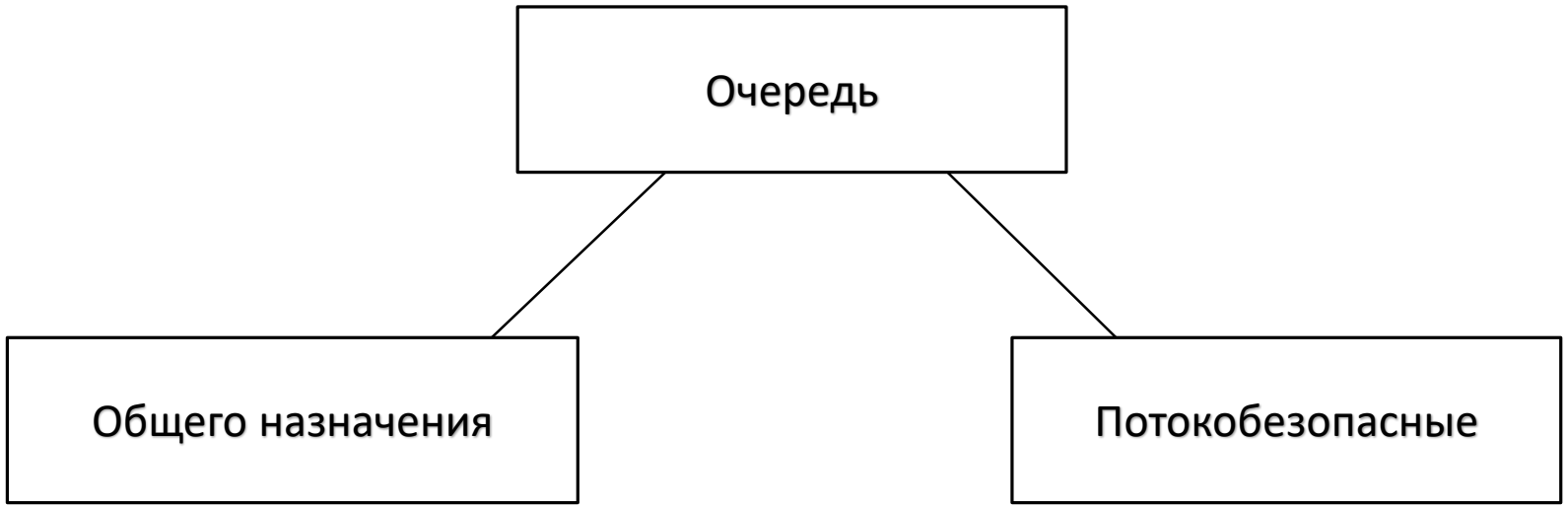
Пример посложнее (результат работы)

```java
public class CustomerMain {
    static protected CountDownLatch latch = new CountDownLatch(2);

    public static void main(String[] args) {
        PriorityQueue<CustomerOrder> queue = new PriorityQueue<>();
        ExecutorService executor;
        executor = Executors.newFixedThreadPool( nThreads: 2);

        System.out.println("Запуск потоков");
        executor.execute(new CustomerGiver(queue));
        executor.execute(new CustomerTaker(queue));

        try {
            latch.await();
        } catch (InterruptedException e) {
            System.out.println("Работа потоков завершена");
        }
        executor.shutdown();
    }
}
```

```
Запуск потоков
В очередь добавлен: Customer name: customer1, order amount: 27.0, order id: 1
Обработка очереди: Customer name: customer1, order amount: 27.0, order id: 1, order price: 270.0
В очередь добавлен: Customer name: customer2, order amount: 13.0, order id: 4
В очередь добавлен: Customer name: customer3, order amount: 13.5, order id: 2
В очередь добавлен: Customer name: customer4, order amount: 4.3, order id: 8
В очередь добавлен: Customer name: customer5, order amount: 2.0, order id: 26
Количество попыток записи в занятую очередь: 1
Обработка очереди: Customer name: customer5, order amount: 2.0, order id: 26, order price: 20.0
В очередь добавлен: Customer name: customer6, order amount: 29.7, order id: 15
Количество попыток записи в занятую очередь: 1
Количество попыток записи в занятую очередь: 2
Количество попыток записи в занятую очередь: 3
Количество попыток записи в занятую очередь: 4
Слишком много попыток записи в занятую очередь.
Обработка очереди: Customer name: customer6, order amount: 29.7, order id: 15, order price: 297.0
Обработка очереди: Customer name: customer4, order amount: 4.3, order id: 8, order price: 43.0
Обработка очереди: Customer name: customer2, order amount: 13.0, order id: 4, order price: 130.0
Обработка очереди: Customer name: customer3, order amount: 13.5, order id: 2, order price: 135.0
Очередь пуста. Прекращение работы.

Process finished with exit code 0
```

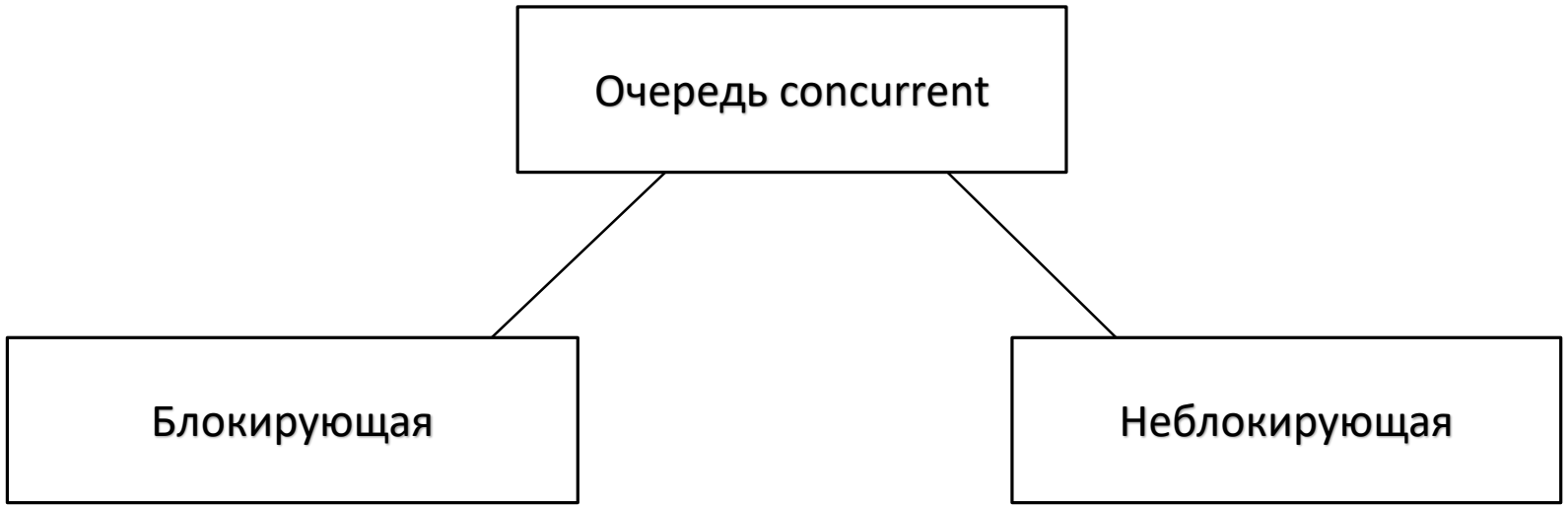
Потокобезопасные очереди

Начиная с JDK 1.5, в Java появились реализации очередей, оптимизированные под использование в многопоточных приложениях.



Потокобезопасные очереди

В свою очередь, потокобезопасные очереди делятся на два типа: блокирующие и неблокирующие.



Потокобезопасные очереди

Разновидности блокирующих очередей:

1) ArrayBlockingQueue
2) LinkedBlockingQueue
3) LinkedBlockingDeque
4) SynchronousQueue
5) LinkedTransferQueue
6) DelayQueue
7) PriorityBlockingQueue

Разновидности неблокирующих очередей:

1) ConcurrentLinkedQueue
2) ConcurrentLinkedDeque

Пример работы с блокирующей очередью

```java
@Override
public void run() {
    try {
        for (int i = 0; i < this.customerList.size(); i++) {
            if (this.queue.size() >= 4) {
                i--;
            } else {
                this.queue.put(this.customerList.get(i));
                System.out.println("В очередь добавлен: " + this.customerList.get(i).toString());
            }
            Thread.sleep( l: 500);
        }
    } catch (InterruptedException e) {
        e.printStackTrace();
    }

    CustomerMainConcurrent.latch.countDown();
}
```

```java
@Override
public void run() {
    CustomerOrderConcurrent order;
    while (true) {
        if (queue.isEmpty()) {
            System.out.println("Очередь пуста. Прекращение работы.");
            CustomerMainConcurrent.latch.countDown();
            break;
        }

        try {
            order = this.queue.take();
            order.setPrice(order.getOrderAmount() * 10d);
            System.out.println("Обработка очереди: " + order.toString() + ", order price: " + order.getPrice());
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        try {
            Thread.sleep( l: 3000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

Пример работы с блокирующей очередью (результат работы)

```java
public class CustomerMainConcurrent {
    static protected CountDownLatch latch = new CountDownLatch(2);

    public static void main(String[] args) {
        PriorityBlockingQueue<CustomerOrderConcurrent> queue = new PriorityBlockingQueue<>( initialCapacity: 4);
        ExecutorService executor;
        executor = Executors.newFixedThreadPool( nThreads: 2);

        System.out.println("Запуск потоков");
        executor.execute(new CustomerGiverConcurrent(queue));
        executor.execute(new CustomerTakerConcurrent(queue));

        try {
            latch.await();
        } catch (InterruptedException e) {
            System.out.println("Работа потоков завершена");
        }
        executor.shutdown();
    }
}
```

```
Запуск потоков
В очередь добавлен: Customer name: customer1, order amount: 27.0, order id: 1
Обработка очереди: Customer name: customer1, order amount: 27.0, order id: 1, order price: 270.0
В очередь добавлен: Customer name: customer2, order amount: 13.0, order id: 4
В очередь добавлен: Customer name: customer3, order amount: 13.5, order id: 2
В очередь добавлен: Customer name: customer4, order amount: 4.3, order id: 8
В очередь добавлен: Customer name: customer5, order amount: 2.0, order id: 26
Обработка очереди: Customer name: customer5, order amount: 2.0, order id: 26, order price: 20.0
В очередь добавлен: Customer name: customer6, order amount: 29.7, order id: 15
Обработка очереди: Customer name: customer6, order amount: 29.7, order id: 15, order price: 297.0
В очередь добавлен: Customer name: customer7, order amount: 5.0, order id: 9
Обработка очереди: Customer name: customer7, order amount: 5.0, order id: 9, order price: 50.0
В очередь добавлен: Customer name: customer8, order amount: 67.3, order id: 2
Обработка очереди: Customer name: customer4, order amount: 4.3, order id: 8, order price: 43.0
В очередь добавлен: Customer name: customer9, order amount: 14.0, order id: 1
Обработка очереди: Customer name: customer2, order amount: 13.0, order id: 4, order price: 130.0
Обработка очереди: Customer name: customer8, order amount: 67.3, order id: 2, order price: 673.0
Обработка очереди: Customer name: customer3, order amount: 13.5, order id: 2, order price: 135.0
Обработка очереди: Customer name: customer9, order amount: 14.0, order id: 1, order price: 140.0
Очередь пуста. Прекращение работы.
```

Интерфейс Deque

Интерфейс Deque расширяет интерфейс Queue и определяет поведение двунаправленной очереди, которая работает как обычная однонаправленная очередь, либо как стек, действующий по принципу LIFO (последний вошел - первый вышел).

Методы интерфейса Deque

- ***void addFirst(E obj):*** добавляет элемент в начало очереди
- ***void addLast(E obj):*** добавляет элемент obj в конец очереди
- ***E getFirst():*** возвращает без удаления элемент из головы очереди. Если очередь пуста, генерирует исключение NoSuchElementException
- ***E getLast():*** возвращает без удаления последний элемент очереди. Если очередь пуста, генерирует исключение NoSuchElementException
- ***boolean offerFirst(E obj):*** добавляет элемент obj в самое начало очереди. Если элемент удачно добавлен, возвращает true, иначе – false
- ***boolean offerLast(E obj):*** добавляет элемент obj в конец очереди. Если элемент удачно добавлен, возвращает true, иначе - false

Методы интерфейса Deque

- ***E peekFirst():*** возвращает без удаления элемент из начала очереди. Если очередь пуста, возвращает значение null
- ***E peekLast():*** возвращает без удаления последний элемент очереди. Если очередь пуста, возвращает значение null
- ***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 pop():*** возвращает с удалением элемент из начала очереди. Если очередь пуста, генерирует исключение NoSuchElementException
- ***void push(E element):*** добавляет элемент в самое начало очереди

Методы интерфейса Deque

- ***E removeFirst():*** возвращает с удалением элемент из начала очереди. Если очередь пуста, генерирует исключение NoSuchElementException
- ***E removeLast():*** возвращает с удалением элемент из конца очереди. Если очередь пуста, генерирует исключение NoSuchElementException
- ***boolean removeFirstOccurrence(Object obj):*** удаляет первый встреченный элемент obj из очереди. Если удаление произшло, то возвращает true, иначе возвращает false.
- ***boolean removeLastOccurrence(Object obj):*** удаляет последний встреченный элемент obj из очереди. Если удаление произшло, то возвращает true, иначе возвращает false.

LinkedList – чудовище Франкенштейна от мира коллекций в Java

```java
public class QueueMain4 {
    public static void main(String[] args) {
        Queue<Integer> queue = new LinkedList<>();
        queue.offer( e: null);
        System.out.println("Созданная очередь: " + queue);
        System.out.println("Голова очереди (метод element): " + queue.element());
    }
}
```

```java
public class QueueMain5 {
    public static void main(String[] args) {
        Queue<Integer> queue = new PriorityQueue<>();
        queue.offer( e: null);
        System.out.println("Созданная очередь: " + queue);
        System.out.println("Голова очереди (метод element): " + queue.element());
    }
}
```

```
Созданная очередь: [null]
Голова очереди (метод element): null

Process finished with exit code 0
```

```
Exception in thread "main" java.lang.NullPointerException Create breakpoint
    at java.base/java.util.PriorityQueue.offer(PriorityQueue.java:340)
    at homeworks.homework7.examples.QueueMain5.main(QueueMain5.java:9)

Process finished with exit code 1
```