



Java course

- [Начало Java](#)
- [Проект «Отдел кадров»](#)
- [Курсы](#)
- [Статьи](#)
- [Контакты/Вопросы](#)

- [Введение](#)
- [Установка JDK](#)
- [Основные шаги](#)
- [Данные](#)
- [Порядок операций](#)
- [IDE NetBeans](#)
- [ООП](#)
- [Инкапсуляция](#)
- [Наследование](#)
- [Пакеты](#)
- [Переопределение и перегрузка](#)
- [Полиморфизм](#)
- [Статические свойства и методы](#)
- [Отношения между классами](#)
- [Визуализация работа](#)
- [Пример — очередь объектов](#)
- [Массивы — знакомство](#)
- [Многомерные массивы](#)
- [Абстрактные классы](#)
- [Интерфейсы](#)

Пример — очередь объектов

В этой части я хочу предложить пример, который позволит нам посмотреть как можно организовать взаимодействие объектов для реализации достаточно распространенной структуры данных — очереди. Т.е. я хочу создать класс, который позволит мне «складывать» туда объекты в заранее неизвестном количестве и «вынимать» объекты из этой очереди. Такая функциональность часто называется FIFO — First In First Out — первый пришел, первый ушел. По сути наш класс должен иметь три метода:

1. Положить объект (произвольного класса) в очередь — назовем метод **push**
2. Вытащить объект (произвольного класса) из очереди — назовем метод **pull**
3. Получить количество объектов в очереди — назовем его **size**

Итак, наш класс **ObjectQueue** будет выглядеть (без реализации, только с методами) вот так:

- [Расширенное описание классов](#)
- [Исключения](#)
- [Решения на основе классов](#)
- [Список контактов — начало](#)
- [Коллекции — базовые принципы](#)
- [Коллекции — продолжение](#)
- [Список контактов — GUI приложение](#)
- [Что такое JAR-файлы](#)
- [Многопоточность — первые шаги](#)
- [Многопоточность и синхронизация](#)
- [Работаем с XML](#)
- [Reflection — основы](#)
- [Установка СУБД PostgreSQL](#)
- [Базы данных на Java — первые шаги](#)
- [Возможности JDBC — второй этап](#)
- [JDBC — групповые операции](#)
- [Список контактов — работаем с БД](#)
- [Переезжаем на Maven](#)
- [Потоки ввода-вывода](#)
- [Сетевое взаимодействие](#)
- [С чего начинается Web](#)

```
1 public class ObjectQueue
2 {
3     public void push(Object obj) {
4         // Пока реализации нет
5     }
6
7     public Object pull() {
8         // Пока реализации нет
9     }
10
11     public int size() {
12         // Пока реализации нет
13     }
14 }
```

Как видите, мы можем положить в очередь объект типа **Object**, что означает, что мы можем положить туда все, что угодно — как вы должны помнить, класс **Object** является «проматерью» всех классов — все классы наследуются в конечном итоге от него (если не прямо от него, то он будет их

дедушкой, прадедушкой и т.д.).

Предлагаю сразу написать маленький класс для тестирования нашей очереди

```
1 public class QueueTest
2 {
3     public static void main(String[] arg) {
4         ObjectQueue queue = new ObjectQueue();
5
6         for(int i=0; i<10; i++) {
7             // В данном случае мы складываем в очередь строки
8             queue.push("Строка:" + i);
9         }
10
11        while(queue.size() > 0) {
12            // Мы делаем жесткое приведение типа, т.к. знаем, что там лежат строки
13            String s = (String)queue.pull();
14            System.out.println(s);
15            System.out.println("Размер очереди:" + queue.size());
16        }
17    }
18 }
```

Можно видеть, что сначала мы помещаем в очередь объект класса **String** (первый цикл от 0 до 10 и потом выбираем все объекты до тех пор, пока размер очереди не станет нулевым. Обратите внимание на то, что мы делаем приведение типа объекта при его получении. Когда мы начнем говорить о коллекциях, мы об этом вспомним.

Теперь давайте подумаем над реализацией нашей очереди. Большинство книг по структурам данных содержит такое понятие как связный список. Если быть точнее — однонаправленный связный список. Сама идея списка достаточно простая — ее можно посмотреть на рисунке.



Как видите, все достаточно несложно. У нас есть структура, которая содержит два элемента:

1. Хранилище для данных
2. Указатель на следующий элемент

При добавлении нового элемента нам надо создать такую структуру (по сути это объект с двумя полями), поместить в хранилище данных переданный объект и указатель у последнего элемента в списке «указать» на добавляемый объект. Мы как бы «прикрепляем» новый объект к существующей цепочке объектов. Однонаправленная очередь потому, что двигаться по ней можно только в одном направлении — от «головы» к «хвосту». Как мы только что говорили, нам потребуется вспомогательный объект — назовем его **ObjectBox** и т.к. он в общем-то никому не нужен, мы можем объявить его внутри нашего класса **ObjectQueue**. Смотрим:

```
1 private class ObjectBox {
2     private Object object;
3     private ObjectBox next;
4
5     public Object getObject() {
6         return object;
7     }
8
9     public void setObject(Object object) {
10        this.object = object;
11    }
12
13    public ObjectBox getNext() {
14        return next;
15    }
16
17    public void setNext(ObjectBox next) {
18        this.next = next;
19    }
20 }
```

Как видите — все просто. В поле **object** мы будем помещать сам добавляемый объект, а поле **next** будет указывать на следующий элемент в цепочке. Теперь поговорим о классе **ObjectQueue**. Ему необходимо иметь поле, которое указывает на самый первый элемент — нам же надо с чего-то начинать и брать элементы из очереди мы будем как раз с «головы». В принципе одного этого элемента достаточно, но это будет крайне не эффективно. Потому что при добавлении вам придется каждый раз пробегать от «головы» до «хвоста» и уже только после нахождения «хвоста» можно будет добавлять новый элемент. В качестве тренировки вы можете реализовать сами такой вариант. Но мы все-таки сделаем более корректно — введем еще одно поле, которое будет всегда указывать на «хвост» — на последний элемент в очереди. Наличие этого элемента позволяет создавать очень эффективную структуру с очень важной характеристикой — время добавления нового элемента не зависит от размера списка. Очередь может содержать 100 элементов или 100000 — время добавления будет всегда одинаковое. Но у такой структуры есть минус — если вы хотите найти элемент на определенной позиции — например 184-й — то эта операция будет исполняться долго для больших списков. Но вернемся к задаче — как обычно, я предлагаю сразу посмотреть код и комментарии к нему.

```
1 package edu.javacourse.queue;
```

```
2
3 public class ObjectQueue
4 {
5     // Указатель на первый элемент
6     private ObjectBox head = null;
7     // Указатель на последний элемент
8     private ObjectBox tail = null;
9     // Поле для хранения размера очереди
10    private int size = 0;
11
12    public void push(Object obj) {
13        // Сразу создаем вспомогательный объект и помещаем новый элемент в него
14        ObjectBox ob = new ObjectBox();
15        ob.setObject(obj);
16        // Если очередь пустая - в ней еще нет элементов
17        if (head == null) {
18            // Теперь наша голова указывает на наш первый элемент
19            head = ob;
20        } else {
21            // Если это не первый элемент, то надо, чтобы последний элемент в очереди
22            // указывал на вновь прибывший элемент
23            tail.setNext(ob);
24        }
25        // И в любом случае нам надо наш "хвост" переместить на новый элемент
26        // Если это первый элемент, то "голова" и "хвост" будут указывать на один и тот же элемент
27        tail = ob;
28        // Увеличиваем размер нашей очереди
29        size++;
30    }
31
32    public Object pull() {
33        // Если у нас нет элементов, то возвращаем null
34        if (size == 0) {
35            return null;
36        }
37        // Получаем наш объект из вспомогательного класса из "головы"
38        Object obj = head.getObject();
39        // Перемещаем "голову" на следующий элемент
40        head = head.getNext();
41        // Если это был единственный элемент, то head станет равен null
42        // и тогда tail (хвост) тоже должен указать на null.
43        if (head == null) {
44            tail = null;
45        }
46        // Уменьшаем размер очереди
47        size--;
48        // Возвращаем значение
```

```

49     return obj;
50 }
51
52 public int size() {
53     return size;
54 }
55
56 // Наш вспомогательный класс будет закрыт от посторонних глаз
57 private class ObjectBox
58 {
59     // Поле для хранения объекта
60     private Object object;
61     // Поле для указания на следующий элемент в цепочке.
62     // Если оно равно NULL - значит это последний элемент
63     private ObjectBox next;
64
65     public Object getObject() {
66         return object;
67     }
68
69     public void setObject(Object object) {
70         this.object = object;
71     }
72
73     public ObjectBox getNext() {
74         return next;
75     }
76
77     public void setNext(ObjectBox next) {
78         this.next = next;
79     }
80 }
81 }

```

Класс для тестирования нашей очереди

```

1 package edu.javacourse.queue;
2
3 public class QueueTest
4 {
5     public static void main(String[] arg) {
6         ObjectQueue queue = new ObjectQueue();
7
8         for(int i=0; i<10; i++) {

```

```

9         // В данном случае мы складываем в очередь строки
10        queue.push("Строка:" + i);
11    }
12
13    while(queue.size() > 0) {
14        // Мы делаем жесткое приведение типа, т.к. знаем, что там лежат строки
15        String s = (String)queue.pull();
16        System.out.println(s);
17        System.out.println("Размер очереди:" + queue.size());
18    }
19 }
20 }

```

Пример — очередь объектов. Добавляем функциональность

Теперь мы добавим удобную функцию — получение элемента по индексу (по номеру в очереди. Эта функция не будет удалять элемент из очереди — она просто вернет элемент. В этом коде мы увидим, что скорость исполнения здесь зависит от размера очереди — чем он больше, тем функция будет работать дольше. И как говорил герой из фильма «В бой идут одни старики» — «Сергея, не надо слов». Смотрим код.

```

1 package edu.javacourse.queue;
2
3 public class ObjectQueue
4 {
5     // Указатель на первый элемент
6     private ObjectBox head = null;
7     // Указатель на последний элемент
8     private ObjectBox tail = null;
9     // Поле для хранения размера очереди
10    private int size = 0;
11
12    public void push(Object obj) {
13        // Сразу создаем вспомогательный объект и помещаем новый элемент в него
14        ObjectBox ob = new ObjectBox();
15        ob.setObject(obj);
16        // Если очередь пустая - в ней еще нет элементов
17        if (head == null) {
18            // Теперь наша голова указывает на наш первый элемент
19            head = ob;
20        } else {
21            // Если это не первый элемент, то надо, чтобы последний элемент в очереди
22            // указывал на вновь прибывший элемент
23            tail.setNext(ob);
24        }

```

```
25 // И в любом случае нам надо наш "хвост" переместить на новый элемент
26 // Если это первый элемент, то "голова" и "хвост" будут указывать на один и тот же элемент
27 tail = ob;
28 // Увеличиваем размер нашей очереди
29 size++;
30 }
31
32 public Object pull() {
33     // Если у нас нет элементов, то возвращаем null
34     if (size == 0) {
35         return null;
36     }
37     // Получаем наш объект из вспомогательного класса из "головы"
38     Object obj = head.getObject();
39     // Перемещаем "голову" на следующий элемент
40     head = head.getNext();
41     // Если это был единственный элемент, то head станет равен null
42     // и тогда tail (хвост) тоже должен указать на null.
43     if (head == null) {
44         tail = null;
45     }
46     // Уменьшаем размер очереди
47     size--;
48     // Возвращаем значение
49     return obj;
50 }
51
52 public Object get(int index) {
53     // Если нет элементов или индекс больше размера или индекс меньше 0
54     if (size == 0 || index >= size || index < 0) {
55         return null;
56     }
57     // Устанавливаем указатель, который будем перемещать на "голову"
58     ObjectBox current = head;
59     // В этом случае позиция равна 0
60     int pos = 0;
61     // Пока позиция не достигла нужного индекса
62     while (pos < index) {
63         // Перемещаемся на следующий элемент
64         current = current.getNext();
65         // И увеличиваем позицию
66         pos++;
67     }
68     // Мы дошли до нужной позиции и теперь можем вернуть элемент
69     Object obj = current.getObject();
70     return obj;
71 }
```



```

72
73     public int size() {
74         return size;
75     }
76
77     // Наш вспомогательный класс будет закрыт от посторонних глаз
78     private class ObjectBox
79     {
80         // Поле для хранения объекта
81         private Object object;
82         // Поле для указания на следующий элемент в цепочке.
83         // Если оно равно NULL - значит это последний элемент
84         private ObjectBox next;
85
86         public Object getObject() {
87             return object;
88         }
89
90         public void setObject(Object object) {
91             this.object = object;
92         }
93
94         public ObjectBox getNext() {
95             return next;
96         }
97
98         public void setNext(ObjectBox next) {
99             this.next = next;
100         }
101     }
102 }

```

Полный код проекта можно скачать отсюда: [ObjectQueue.zip](#).

Домашнее задание

Теперь можно (и нужно) поработать самим — предлагаю несколько вариантов самостоятельной работы:

1. Реализовать двунаправленный список — можно двигаться не только от головы к хвосту, но и от хвоста к голове. Это позволит ускорить выполнение функции **get**. В качестве подсказки — теперь класс **ObjectBox** должен иметь не только поле **next**, но и поле **prev**, которое должно указывать на предыдущий элемент в списке
2. Реализовать не очередь, а стек — LIFO — Last In First Out (последний пришел, первый ушел). Что-то вроде нанизывания колец на столб — можно снять только последнее
3. Реализовать раздел [Визуализация робота](#) с помощью нашей очереди

4. Реализовать очередь с приоритетами — при добавлении элемента вы можете указать его приоритетность (от 1 до 10). В этом случае вам надо вставить элемент не в самый конец, а после элемента с таким же приоритетом (или с большим — если элементов с таким же приоритетом нет)

Удачи.

И теперь нас ждет следующая статья: [Массивы — знакомство](#).

77 comments to *Пример — очередь объектов*



Февраль 14, 2016 at 14:14

Алекс says:

Уважаемый автор, мне также почему-то тяжело въехать в реализацию механизма «//Перемещаем «голову» на следующий элемент».

Насколько я понимаю, в данном участке кода метода pull() два указанных метода выполняются одним и тем же объектом класса ObjectBox (например, объектом ob1).

// Получаем наш объект из вспомогательного класса из «головы»

Object obj = head.getObject();

// Перемещаем «голову» на следующий элемент

head = head.getNext();

// Если это был единственный элемент, то head станет равен null

// и тогда tail (хвост) тоже должен указать на null.

Но тогда каким образом указатель head перемещается на следующий объект (например, ob2)? Это связано со спецификой хранения и обработки данных в памяти (типа после выполнения метода head.getObject() этот объект стирается?) или может быть next это какой-то оператор?

И могли бы Вы прояснить еще один вопрос: каким образом заполняется поле next для первого объекта класса ObjectBox в методе push()(если объектов больше одного), когда участок кода

```
else {
```

```
// Если это не первый элемент, то надо, чтобы последний элемент в очереди
```

```
// указывал на вновь прибывший элемент
```

```
tail.setNext(ob);
```

```
}
```

для первого объекта не выполняется, т.к. правдиво условие if (head == null) ?

[Reply](#)



o

Февраль 26, 2016 at 15:56

admin says:

По вопросу: «Но тогда каким образом указатель head перемещается на следующий объект»

Так вот же код:

```
head = head.getNext();
```

Голова теперь «переехала» на следующий элемент. А бывший головной элемент теперь уже недоступен — он находится только в переменной `obj`, которую мы и отдаем.

По вопросу: «каким образом заполняется поле `next` для первого объекта класса `ObjectBox` в методе `push()`»

Она равна `null` — если элемент первый и единственный — значит за ним никого нет. И это обозначается `null`. И `tail` тогда указывает на тот же элемент, что и `head`. Когда же мы добавляем еще один элемент (их становится больше одного), тогда `tail` будет сдвигаться к последнему пришедшему элементу — это решается кодом:

```
tail.setNext(obj); // Теперь последний элемент будет указывать на вновь пришедший и он станет не последним, последним будет вновь прибывший элемент.
```

```
tail = obj; // И теперь и tail указывает на последний элемент (который только что пришел)
```

А `head` никуда не сдвигается — он так и указывает на первый элемент.

Таким образом получаем — `head` всегда указывает на первый элемент, а `tail` — на последний.

[Reply](#)



o

Апрель 20, 2016 at 10:44

Арте́м says:

По вопросу: «каким образом заполняется поле `next` для первого объекта класса `ObjectBox` в методе `push()`» как и сказал уважаемый `admin` поле `next` равно `null` — если элемент первый и единственный, кроме того ссылки `head` и `tail` указывают на один и тот же объект — `ob`. При дальнейшем добавлении объектов в очередь происходит создание нового объекта, `ob` (но это уже не тот объект `ob`, который создавался для первого элемента, мысленно назовем его `ob1`), идет запись в него новой строки, а затем после выполнения `tail.setNext(ob)`; этот новый объект `ob1` записывается в поле `next` объекта `ob` на который ссылается как `tail` так и `head`. Следующей строкой `tail = ob`; ссылка `tail` обновляется (в ней теперь находится только `ob1`). По ссылке `head` имеем теперь первый объект `ob` в поле `next` которого находится новый объект `ob1`. При дальнейшем добавлении объектов в очередь создается новый объект `ob2` с новой строкой, после выполнения

`tail.setNext(ob);` идет его запись в поле `next` объекта `ob1`, а затем ссылка `tail` снова обновляется и указывает лишь на последний объект `ob2`. Таким образом по ссылке `head` теперь находится первый объект `ob` в поле `next` которого находится новый объект `ob1` в поле `next` которого находится `ob2`. При дальнейшем добавлении объектов в очередь по ссылке `head` будут доступны все элементы очереди объектов (как матрешки), а `tail` всегда будет указывать на последний. Прошу прощения за сумбурное изложение, пытался объяснить как сам понял.

[Reply](#)



Июль 30, 2017 at 21:38

Nokola says:

Никак не въеду в это:»... после выполнения `tail.setNext(ob);` этот новый объект `ob1` записывается в поле `next` объекта `ob...`. Объясните, пожалуйста, как эта «магия» происходит?

[Reply](#)



Июль 31, 2017 at 11:44

admin says:

Магия заключается в том, что выстраивается цепочка. Еще раз смотрим на слово — «цепочка». В ней объекты «цепляются» друг за друга. Первый указывает на второй, второй — на третий и т.д. Когда добавляется новый объект, он помещается в конец цепочки. Для того, чтобы последний объект в цепочке стал предпоследним, он должен указать на «новенького» — на добавленный объект. Вы как бы подключаете новое звено в цепочку — цепляете новое звено к последнему в цепи.

[Reply](#)



Август 1, 2017 at 15:00

Nokola says:

Спасибо уважаемый админ, но легче не стало:-). Что такое структура по принципу цепочки я понял. Вопрос в том, почему после выполнения `tail.setNext(ob)` (предположим что элемент в цепочке не первый и `tail` не равен `null`, а хранит в

себе значение предыдущего элемента `ob`) происходит присваивание значения текущего элемента `ob` именно полю `next` предыдущего элемента?



■ Август 2, 2017 at 09:53

admin says:

Здесь уже обычным ответом сложно обойтись — надо долго и образно объяснять. Такое бывает, но я вряд ли смогу как-то еще на словах без рисунков, жестов и интонаций что-то объяснить 😊

Надо читать код и одновременно видеть «картинку» объектов, которые выстраиваются в цепочку.

Вот именно по этой причине нужны занятия с преподавателем. Попробуйте еще раз «прочитать» код и нарисовать взаимодействие объектов (прямо на бумаге нарисовать).



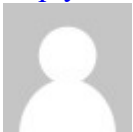
• Апрель 16, 2016 at 21:30

Ilya says:

Спасибо автору за очередную прекрасную статью.

Есть 1 пожелание. Вы можете выложить правильный текст классов для «вариантов для самостоятельной работы»? Чтобы мы смогли проверить себя/найти ошибку или на худой конец, если ничего не получается, то прочитать, запомнить и осознать и эти варианты.

[Reply](#)



• Апрель 17, 2016 at 18:08

Artem says:

`tail.setNext(ob); /* Можно ли сказать, что в этой строке в поле next объекта tail идет запись объекта ob? Или tail здесь не объект, а переменная типа ObjectBox? */`

[Reply](#)



Апрель 23, 2016 at 11:52

Арте́м says:

Мой вариант решения первой задачи(двунаправленный список). Основная сложность заключалась в строчке `ob.setPrev(tail);` метода `push()`.

```
public void push(Object obj){  
    ObjectBox ob = new ObjectBox();  
    ob.setObject(obj);
```

```
    if(head==null)  
    {  
        head = ob;  
    }  
    else  
    {  
        tail.setNext(ob);  
    }
```

```
    ob.setPrev(tail);  
    tail=ob;
```

```
    size++;
```

```
    }  
    public Object getIndex(int index){
```

```
        if(size==0||size<=index||index<0)  
            return null;
```

```
        if(index<size/2)  
        {  
            ObjectBox current = head;  
            int pos = 0;  
            while(pos<index)  
            {  
                current = current.getPrev();
```

```
pos—;  
}  
Object obj = current.getObject();  
return obj;  
}  
}
```

[Reply.](#)



• Апрель 23, 2016 at 11:56

Арте́м says:

Почему то весь код не скопировался((

[Reply.](#)



o

Апрель 25, 2016 at 10:00

admin says:

Заклучайте весь код в тэг <pre> ... </pre>

[Reply.](#)



• Апрель 25, 2016 at 12:07

Арте́м says:

```
1 public void push(Object obj) {  
2     ObjectBox ob = new ObjectBox();  
3     ob.setObject(obj);  
4  
5     if(head==null)
```

```

6      {
7          head = ob;
8      }
9      else
10     {
11         tail.setNext(ob);
12     }
13
14     ob.setPrev(tail);
15     tail=ob;
16
17     size++;
18 }
19 public Object getIndex(int index) {
20
21     if(size==0||size<=index||index<0)
22         return null;
23
24     if(index<size/2)
25     {
26         ObjectBox current = head;
27         int pos = 0;
28         while(pos<index)
29         {
30             current = current.getPrev();
31             pos--;
32         }
33         Object obj = current.getObject();
34         return obj;
35     }
36 }

```

[Reply.](#)



Апрель 25, 2016 at 12:11

Арте́м says:

Все равно метод getIndex скопировался не полностью, даю его в отдельном сообщении:

```

1 public Object getIndex(int index) {

```



```
2
3     if(size==0||size<=index||index<0)
4         return null;
5
6     if(index<size/2)
7     {
8         ObjectBox current = head;
9         int pos = 0;
10        while(pos<index)
11        {
12            current = current.getPrev();
13            pos--;
14        }
15        Object obj = current.getObject();
16        return obj;
17    }
18 }
```

[Reply.](#)



• Апрель 25, 2016 at 12:13

Арте́м says:

Все равно не копирует!!

[Reply.](#)



o

Апрель 25, 2016 at 12:34

admin says:

А что не видно ?

[Reply.](#)



•

Апрель 25, 2016 at 16:28

Арте́м says:

Не видно конца условия if и начала второго условия else. Попробую дать по частям, и попрошу убрать не удавшиеся комментарии.

[Reply.](#)



Апрель 25, 2016 at 16:30

Арте́м says:

```
1 public Object getIndex(int index) {
2
3     if(size==0||size<=index||index<0)
4         return null;
5
6     if(index<size/2)
7     {
8         ObjectBox current = head;
9         int pos = 0;
10        while(pos<index)
11        {
12            current = current.getNext();
13            pos++;
14        }
15        Object obj = current.getObject();
16        return obj;
17    }
```

[Reply.](#)



Апрель 25, 2016 at 16:31

Арте́м says:

```

1  else
2      {
3          ObjectBox current = tail;
4          int pos = size-1;
5          while(pos>index)
6              {
7                  current = current.getPrev();
8                  pos--;
9              }
10         Object obj = current.getObject();
11         return obj;
12     }
13 }

```

[Reply](#)



• Апрель 25, 2016 at 16:32

Арте́м says:

По частям получилось!

[Reply](#)



• Апрель 25, 2016 at 16:36

Арте́м says:

По второму заданию — «Реализовать не очередь, а стек — LIFO — Last In First Out (последний пришел, первый ушел)». Если получилось сделать первое, то второе — «the peace of cake»))

```

1  public Object pullStack() {
2      if (size == 0)
3          return null;
4      Object obj = tail.getObject();
5      tail = tail.getPrev();
6      if (head == null)

```

```
7 |         tail = null;
8 |         size--;
9 |         return obj;
10 |     }
```

[Reply.](#)



Апрель 25, 2016 at 16:46

Artem says:

В смысле ріесе конечно))))

[Reply.](#)



Апрель 26, 2016 at 09:28

admin says:

Лучше такого рода вопросы на почту отправлять — комментировать код здесь достаточно неудобно.

[Reply.](#)



Май 31, 2016 at 15:36

Nibbler says:

Прошу прощения, но я никак не могу осознать конструкцию:

```
1 | tail.setNext(obj);
```

Как я понял — `tail` — переменная с типом `ObjectBox`, которая, по идее, должна бы ссылаться на новый объект класса `ObjectBox`, если бы мы его создали: `ObjectBox tail = new ObjectBox()`. Каким образом мы применяем к этой переменной, которая инициализирована `null`-ом в самом начале,

метод без создания объекта? Затем, методу отдаем только что созданный объект `ObjectBox`.

До этого в курсе я подобных «фишек» не видел. Или, может быть, я где-то что-то по невнимательности пропустил... 😞

[Reply](#)



Май 31, 2016 at 21:46

Nibbler says:

Предыдущий вопрос снимаю. Осознал, пока ехал в электричке — к тому времени, когда мы вызываем метод `setNext` переменная `tail` уже содержит указатель на объект. Это — предыдущий объект `ob`, переданный ей присвоением `tail = ob`; которое я поначалу не заметил.

[Reply](#)



Июнь 1, 2016 at 16:44

Nibbler says:

Следуя подсказке (спасибо!), в класс `ObjectBox` добавил новую переменную `private ObjectBox prev`, и соответственно — геттер и сеттер для нее. Сложнее всего было сцепить «вагончики» второй сцепкой 😊 В методе `push` добавил строчку:

```
1 // Если это не первый элемент, то надо, чтобы последний элемент
2 // указывал на вновь прибывший элемент
3 tail.setNext(ob);
4 // а вновь прибывший - указывал на предыдущий
5 ob.setPrev(tail);
```

Дальше — все, вроде бы, просто — от противного. В классе `ObjectQueue` появился дополнительный метод, с помощью которого реализуется выборка «с хвоста»:

```
1 public Object pullback() {
2     // Если у нас нет элементов, то возвращаем null
3     if (size == 0) {
4         return null;
```

```

5      }
6      // Получаем наш объект из вспомогательного класса из "хвоста"
7      Object obj = tail.getObject();
8      // Перемещаем "хвост" на предыдущий элемент
9      tail = tail.getPrev();
10     // Если это был единственный элемент, то tail станет равен null
11     // и тогда head (голова) тоже должна указывать на null.
12     if (tail == null) {
13         head = null;
14     }
15     // Уменьшаем размер очереди
16     size--;
17     // Возвращаем значение
18     return obj;
19 }

```

Метод `get` разбил по условию `if(index < size/2)` на 2 половинки: Если `true` — то поиск нужного индекса идет с головы, если `false` — с хвоста. Спасибо за это занятие — очень "сдвигает парадигму".

Можно вопрос?

Насколько я себе это могу представить, в каждый отдельный момент времени существует только одна ссылка по которой мы сослаться на конкретный объект, в зависимости от состояния, куда мы ее сдвинули. Тем не менее, остальные 9 объектов (в нашем случае их 10 всего), никуда не исчезают и размещены по зарезервированным для них ранее адресам памяти. Может ли случиться такое, что какие-то из этих объектов будут удалены `garbage`-коллектором, если к ним долгое время не будет обращений? Допустим мы выбираем всегда элементы перед 5-м или после 5-го (с головы или с хвоста) — в результате этого перебора указатель ни разу не попадает на 5 элемент — программа его "не дергает" и он спокойно может исчезнуть.

[Reply](#)



o

Июнь 2, 2016 at 17:52

admin says:

Если на объект есть хоть одна ссылка, то он не удаляется. В нашем случае эти ссылки есть в нашем списке. Так что не удаляется.

[Reply](#)



•

Июнь 2, 2016 at 17:25

Nibbler says:

Отрисовка перемещений робота с помощью очереди: Изменения в 2-х классах.

Robot.java: 1) вместо ArrayList создаем объект — очередь 2) в методе forward проталкиваем новую линию в очередь: `queue.push(...)` 3) добавляем геттер `getQueue()`

RobotPathComponent.java: 1) получаем очередь из робота `ObjectQueue oq = robot.getQueue();` 2) в цикле «тянем» линии из очереди: `RobotLine rl = (RobotLine)oq.pull();`

По субъективным ощущениям рисует быстрее, чем раньше (с реализацией в виде списка).

[Reply](#)



o

Июнь 2, 2016 at 17:53

admin says:

Ощущения не самый лучший способ определить скорость — для измерения производительности надо использовать специальный софт.

[Reply](#)



•

Июнь 29, 2016 at 12:46

Firefly says:

Добрый день! Изложение и задания доступны и понятны! Спасибо!

Вопросик у меня: если мы приписываем классу `ObjectBox` модификатор `private`, то не возникнет ли ошибка доступа к нему из класса `ObjectQueue`?

```
1 // Наш вспомогательный класс будет закрыт от посторонних глаз
2 private class ObjectBox
3 {
```

[Reply](#)



o

Июнь 29, 2016 at 14:20

admin says:

Код работает — значит класс доступен. Но только в классе `ObjectQueue`. Можете проверить.

[Reply](#)



■

Июнь 30, 2016 at 00:02

Firefly says:

Наверно дошло: класс `ObjectBox` является внутренним для класса `ObjectQueue`?

Не помню, говорилось ли о таком варианте в ваших уроках, или где-то когда-то уже успела почитать и почти забыть... Я эти классы устраивала в разных файлах!

[Reply](#)



•

Июль 15, 2016 at 15:21

Сергей says:

Опечатка в фразе «При добавлении новґго элемента нам надо создать такую структуру»

[Reply](#)



o

Июль 15, 2016 at 15:52

admin says:

Спасибо, исправил

[Reply](#)



Июль 18, 2016 at 12:01

Сергей says:

Антон, а есть ли практический смысл для экономии системных ресурсов (например для систем с большим количеством пользователей) не ждать, когда «чистильщик» уберет объекты, на которые никто не ссылается поле отработки метода pull(), а в программном коде прописывать «обнуление» объектов очереди командой типа =null ?

[Reply](#)



Июль 18, 2016 at 12:21

admin says:

В том виде, который предлагается — это бессмысленно. Память под объект все равно выделена в куче и ее все равно будет чистить GC. В чем тогда смысл присваивания null ?

Можно вызывать GC самому, но это как раз будет невыгодно — GC требует ресурсов. Настройка работы самого GC — то достаточно деликатная и кропотливая работа.

[Reply](#)



Сентябрь 23, 2016 at 21:48

[Антон](#) says:

Мне пока что в корне не понятна конструкция Object object.

Это что значит? Мы вызываем класс Object (как вы упомянули, являющийся прородителем для всех), называем это типом переменной и далее имя переменной object. В результате получается поле, которое может принять абсолютно любые значения? Зачем вообще это делать?

[Reply](#)



o

Октябрь 3, 2016 at 09:42

admin says:

Чтобы принять объект ЛЮБОГО класса. Потому как все классы в конечном итоге наследники класса `Object`. Значит эта ссылка становится по сути универсальным указателем — она может указывать на любой объект. Что нам и нужно.

[Reply](#)



•

Сентябрь 30, 2016 at 19:50

Антонио says:

Здравствуйте, опишите пожалуйста немного подробнее суть третьего задания, если не трудно!)

[Reply](#)



o

Октябрь 3, 2016 at 17:59

admin says:

Мы сделали хранение списка точек движения робота с помощью стандартного компонента. Я предлагаю сделать это с помощью нашего самописного.

[Reply](#)



•

Сентябрь 30, 2016 at 19:51

Антонио says:

А так вообще круто, с Вашими лекциями за 3 недели уже до этого места дошёл, метод спирали очень действенный)

[Reply](#)



o

Октябрь 3, 2016 at 09:46

admin says:

Спасибо.

[Reply](#)



•

Октябрь 13, 2016 at 20:42

Антонио says:

```
public void push(Object obj, int a) {  
    ObjectBox ob = new ObjectBox();
```

```
    ob.setPrioritet(a);  
    ob.setObject(obj);
```

```
    if (head == null) {  
        head = ob;  
        tail=ob;  
    }  
    else {  
        for (int i=0;i<size;i++){  
            if(ob.getPrioritet()<=tail.getPrioritet()){  
                tail.setNext(ob);  
                ob.setPrev(tail);  
                tail=ob;  
                break;  
            }  
            else {  
                ObjectBox b= tail.getPrev();  
                b.setNext(ob);  
                ob.setPrev(b);
```

```
ob.setNext(tail);
tail.setPrev(ob);

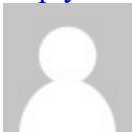
}
}

}

size++;

}
```

[Reply](#)



• Октябрь 13, 2016 at 20:46

Антонио says:

Помогите пожалуйста, сижу над последним заданием этой темы уже несколько дней, ругается начиная со строки `b.setNext(ob)`. По-другому не могу придумать, как всунуть в очередь элемент с большим приоритетом перед существующим

[Reply](#)



• Октябрь 13, 2016 at 20:47

Антонио says:

В компиляторе
Exception in thread «main» java.lang.NullPointerException
at edu.javacourse.queue.ObjectQueue.push(ObjectQueue.java:51)
at edu.javacourse.queue.TestQueue.main(TestQueue.java:22)
Java Result: 1
СБОРКА УСПЕШНО ЗАВЕР

[Reply](#)



• Октябрь 13, 2016 at 20:52

Антонио says:

а как вы переносите красивый код в коменты?

[Reply](#)



о

Октябрь 14, 2016 at 09:33

admin says:

Если нужна помощь с кодом, то лучше присылать на почту — webinar@java-course.ru

[Reply](#)



• Октябрь 24, 2016 at 20:37

Антонио says:

Высылал на указанную почту код примерно неделю назад, ответа до сих пор так и не поступило. Помогите пожалуйста.

[Reply](#)



о

Октябрь 31, 2016 at 16:13

admin says:

Я пока не в состоянии ответить — наберитесь терпения.

[Reply](#)



Ноябрь 3, 2016 at 09:16

v says:

зачем делать жесткую привязку к строке?

```
1 queue.push("John");
2     queue.push("Cat");
3     queue.push(2);
4     queue.push(2.222);
5     queue.push(true);
6     while (queue.size() > 0) {
7         Object s = queue.pull();
8         System.out.println(s);
9         System.out.println("queue size: " + queue.size());
10    }
```

[Reply](#)



Ноябрь 3, 2016 at 10:50

v says:

по своей сути метод size() с возвратом является геттером getSize().

[Reply](#)



Февраль 27, 2017 at 21:16

Юрий says:

Я не постесняюсь и задам самый глупый вопрос. Что нужно сделать в этом задании?)

Хотя бы первый пункт можно разъяснить, что надо сделать?

Второй день сижу медитирую на этот код. Все что я вижу, это то, что в консоли выдается вот такой результат:

Строка: 0

Размер очереди: 9

Строка: 1

Размер очереди: 8

Строка: 2

Размер очереди: 7

Строка: 3

Размер очереди: 6

Строка: 4

Размер очереди: 5

Строка: 5

Размер очереди: 4

Строка: 6

Размер очереди: 3

Строка: 7

Размер очереди: 2

Строка: 8

Размер очереди: 1

Строка: 9

Размер очереди: 0

надо сделать, чтобы цифры не с нуля до девяти а с девяти до нуля, так что ли?

Так это делается просто вот в этой строчке `for (int i = 0 ; i < 10; i++)`

Для чего такой огород нагородили из разных классов?

Может быть действительно пора бы уже выложить правильный код с ответом? А то реально вообще не понятно что надо сделать. Я наверное самый тупой ученик здесь)

[Reply](#)



o

Февраль 28, 2017 at 11:23

admin says:

Вы не могли бы более точно сформулировать — а что Вы делаете (пытаетесь сделать) ?

Кода нет, постановки задачи нет — я не экстрасенс, чтобы догадываться, что за задачу Вы решаете и почему она у Вас не получается 😊

[Reply](#)



Март 1, 2017 at 10:05

Юрий says:

Я пытаюсь понять что нужно сделать в первом задании:) Я более-менее разобрался, как работает та программа, что дана в примере. Но что требуется вот здесь я не могу понять:

«Домашнее задание

Теперь можно (и нужно) поработать самим — предлагаю несколько вариантов самостоятельной работы:

Реализовать двунаправленный список — можно двигаться не только от головы к хвосту, но и от хвоста к голове. Это позволит ускорить выполнение функции get. В качестве подсказки — теперь класс ObjectBox должен иметь не только поле next, но и поле prev, которое должно указывать на предыдущий элемент в списке

»

Что значит двунаправленный список? Как мне понять, что я реализовал эту задачу? Что я должен увидеть на выходе в консоли?

Заранее спасибо

[Reply](#)



o

Март 1, 2017 at 10:18

admin says:

В статье вы можете двигаться только в одну сторону — от первого элемента ко второму, от второго — к третьему и т.д. Вы не можете двигаться от конца к началу — только от начала к концу. Это однонаправленный список — список в одном направлении. Т.е. для вывода последнего элемента надо пройти от начала и до самого конца. При количестве элементов 1 млн. — это долго.

А что увидеть на экране — это уже решайте сами. Или Вы предлагаете мне заниматься Вашим обучением в индивидуальном порядке ? 😊

[Reply](#)



Апрель 12, 2017 at 15:45

Сергей says:

Юрий, надо добавить поле «private ObjectBox prev» классу ObjectBox и реализовать метод push таким образом, чтобы при добавлении экземпляра в очередь, ссылка next предыдущего экземпляра указывала на вновь добавленный, и ссылка prev у вновь добавленного указывала на предыдущий.

```
1  class ObjectBox {
2
3      private Object object;
4      private ObjectBox prev;
5      private ObjectBox next;
6
7      .....
```

Затем, лично я, изменил метод get в классе ObjectQueue так, чтобы в случае, когда мы передаем методу index больше, size / 2 (то есть нам нужен элемент со второй половины ArrayList), то поиск элемента начинается с конца

```
1  public Object get(int index) {
2
3      Object obj = new Object ();
4
5      if (size == 0 || index >= size || index < 0) {
6          return null;
7      }
8
9      if (index <= (size / 2)) {
10         ObjectBox current = head;
11         int pos = 0;
12         while (pos <= (size / 2)) {
13             ObjectBox current = tail;
14             int pos = size - 1;
15             while (pos > index) {
16                 current = current.getPrev ();
17                 pos--;
18             }
19             return current.getObject ();
20         } else {
21             return tail.getObject ();
22         }
23     }
```

Не претендую на правильность и уж тем более качество кода, так как сам три недели, как начал изучать джаву. Но я сделал это задание вот таким вот способом.

[Reply](#)



Апрель 12, 2017 at 15:48

Сергей says:

Почему-то неправильно вставился код метода get. Попробую еще раз

```
1 public Object get(int index) {
2     if (size == 0 || index >= size || index < 0) {
3         return null;
4     }
5     if (index <= (size / 2)) {
6         ObjectBox current = head;
7         int pos = 0;
8         while (pos <= (size / 2)) {
9             ObjectBox current = tail;
10            int pos = size - 1;
11            while (pos > index) {
12                current = current.getPrev ();
13                pos--;
14            }
15            return current.getObject ();
16        } else {
17            return tail.getObject ();
18        }
19    }
```

[Reply](#)



Апрель 12, 2017 at 15:50

Сергей says:

Проблема к админу — удалить код метода `get` из первого моего сообщения и из второго. Почему-то они криво вставляются. То ли по длине сообщения не проходит, то ли еще какие-то проблемы.

[Reply](#)



Май 10, 2017 at 14:53

Sasha says:

Ребят, если интересно по поводу как добавить приоритет в собственную коллекцию:

```
1 public void push(Object object, int priority) {
2     ObjectBox ob = new ObjectBox();
3     // добавляем объекту класса ObjectBox приоритет. Т.е. класс ObjectBox следует доработать
4     ob.setObject(object, priority);
5     if(head==null) {
6         // при пустой коллекции с башкой и хвостом все понятно
7         head = ob;
8         tail = ob;
9     }else {
10         //проверяем, новый элемент должен ли быть добавлен в начало
11         if(ob.getPriority()==tail.getPriority()) {
12             ob.setPrev(tail);
13             tail.setNext(ob);
14             tail=ob;
15         }else {
16             //далее, поняв, что элемент должен быть помещен в середину
17             ObjectBox prev = head;
18             ObjectBox next = head.getNext();
19             //проверяем приоритеты
20             while(!(ob.getPriority()>=prev.getPriority() && ob.getPriority()<next.getPriority())) {
21                 prev = next;
22                 next = prev.getNext();
23             }
24             //переписываем ссылки
25             prev.setNext(ob);
26             ob.setPrev(prev);
27             ob.setNext(next);
28             next.setPrev(ob);
29         }
30     }
```

```
31 |         size++;  
32 |     }
```

[Reply](#)



• Май 10, 2017 at 15:01

Sasha says:

Странно код добавился, почему-то одно условие вообще вылетело при добавлении и изменился оператор сравнения

```
public void push(Object object, int priority) {  
    ObjectBox ob = new ObjectBox();  
    ob.setObject(object, priority);  
    if(head==null) {  
        head = ob;  
        tail = ob;  
    }else {  
        if(ob.getPriority()==tail.getPriority()) {  
            ob.setPrev(tail);  
            tail.setNext(ob);  
            tail=ob;  
        }else {  
            ObjectBox prev = head;  
            ObjectBox next = head.getNext();  
            while(!(ob.getPriority()>=prev.getPriority() && ob.getPriority()<next.getPriority())) {  
                prev = next;  
                next = prev.getNext();  
            }  
            prev.setNext(ob);  
            ob.setPrev(prev);  
            ob.setNext(next);  
            next.setPrev(ob);  
        }  
    }  
    size++;  
}
```

[Reply](#)



• Январь 4, 2018 at 22:08

javazitz says:

Здравствуйте.

А если я хочу создать стек на основе массива, и не знаю заранее в каком кол-ве буду складывать туда объекты, какую размерность выбрать? Или есть какой-нибудь способ менять размерность?

[Reply](#)



○ Январь 5, 2018 at 22:25

admin says:

Можете. Но размерность придется увеличивать. Можно создать новый массив большего размера и скопировать туда старый.

Такая реализация кстати существует. И она эффективно работает в случае, если вы можете представить максимальное количество элементов.

Если же приходится постоянно увеличивать размер — это будет работать плохо.

[Reply](#)



• Январь 29, 2018 at 19:25

MetallFlame says:

Доброго времени суток! У меня возникла проблема при выполнении домашнего задания...

```
public class RobotPathComponent extends JComponent
{
    private Robot robot;
```

```
public RobotPathComponent(Robot robot) {
    this.robot = robot;
```

```

}

@Override
protected void paintComponent(Graphics g) {
    super.paintComponent(g);
    RobotLine line = new RobotLine();
    while (robot.list.size() != 0)
    {
        line = (RobotLine) robot.list.pull();
        int x1 = (int) Math.round(line.getX1());
        int y1 = (int) Math.round(line.getY1());
        int x2 = (int) Math.round(line.getX2());
        int y2 = (int) Math.round(line.getY2());
        g.drawLine(x1, y1, x2, y2); // Линия НЕ рисуется

        /*
        Реализовал очередь объектов для robot, robot.line - объект класса ObjectQueue, который хранит в себе объекты класса RobotLine.
        Линия внутри while не рисуется, хотя в отладчике показано, что все переменные присутствуют, ничего не теряется, все координаты передаются
        в метод drawline.

        */
    }
    g.drawLine(20, 220, 20, 350); // Линия рисуется
}

}

```

[Reply](#)



o

Январь 30, 2018 at 11:03

admin says:

А почему Вы УДАЛЯЕТЕ список линий при рисовании ? Его надо хранить, чтобы при каждой перерисовке можно было бы восстанавливать.

Ведь перерисовка происходит постоянно при изменении ситуации на экране. А Вы уже все удалили — так точно не будет работать.

[Reply](#)



■

Январь 30, 2018 at 14:53

MetallFlame says:

Спасибо большое, мне не хватало понимания этого момента, я думал, что всё рисуется единожды. Получается, в моей ситуации в классе очереди можно написать метод, который просто будет считывать элементы очереди один за одним, не удаляя, и тогда всё будет работать?

[Reply](#)



■ Январь 30, 2018 at 16:52
admin says:

Скорее всего так. Если не получится — лучше прислать все исходники на почту — webinar@java-course.ru. Постараюсь помочь.

[Reply](#)



• Январь 31, 2018 at 14:48
MetallFlame says:

Я решил свою проблемку вот таким способом. Может и не совсем красиво, но в итоге весь путь остается на месте в объекте очереди после прорисовки, всё рисуется. Спасибо за подсказку!

```
1 public class RobotPathComponent extends JComponent
2 {
3     private Robot robot;
4
5     public RobotPathComponent(Robot robot) {
6         this.robot = robot;
7     }
8
9     @Override
10    protected void paintComponent(Graphics g) {
11        super.paintComponent(g);
12        RobotLine line = new RobotLine();
13        for (int i = 0; i < robot.list.size(); i++)
```

```

14     {
15         line = (RobotLine) robot.list.pull();
16         int x1 = (int) Math.round(line.getX1());
17         int y1 = (int) Math.round(line.getY1());
18         int x2 = (int) Math.round(line.getX2());
19         int y2 = (int) Math.round(line.getY2());
20         g.drawLine(x1, y1, x2, y2);
21         robot.list.push(line);
22     }
23     // g.drawLine(20, 220, 20, 350);
24 }
25
26 }

```

[Reply](#)



Февраль 24, 2018 at 14:51

MadFisher says:

Приоритеты. Начал с простого — однонаправленный список. В класс `ObjectBox` добавил поле `int prior`, а также геттер и сеттер для него. вот код метода `push`:

```

1 public void push(Object obj, int prior) {
2
3     ObjectBox ob = new ObjectBox();
4     ob.setObject(obj);
5     ob.setPrior(prior);
6     // Если очередь пустая - в ней еще нет элементов
7     if (head == null) {
8         // Теперь наша голова указывает на наш первый элемент
9         head = ob;
10    } else if (head.getPrior() < prior) {
11        ObjectBox current = head;
12        while (current.getNext() != null && current.getNext().getPrior() >= prior) {
13            current = current.getNext();
14        }
15        ob.setNext(current.getNext());
16        current.setNext(ob);
17    }
18 }

```



```
19 |
20 |         // Увеличиваем размер нашей очереди
21 |         size++;
22 |     }
```

[Reply](#)



o

Февраль 24, 2018 at 19:55

admin says:

А что хочется ? Чтобы я текст программы прокомментировал ? Ну как минимум в else if странный код — там должно быть условие, а не присваивание.

[Reply](#)



■

Февраль 25, 2018 at 08:25

MadFisher says:

Код криво скопировался, в else if пропало условие, и пропал еще один else if.

[Reply](#)



■

Февраль 25, 2018 at 13:45

admin says:

Если хочется проверить код, то лучше присылать на почту — webinar@java-course.ru

[Reply](#)



•

Февраль 24, 2018 at 15:07

MadFisher says:

Я так понимаю проблема с тэгами, вырезано все от знаков меньше-больше

[Reply](#)



o

Февраль 24, 2018 at 19:53

admin says:

Можно все поместить в теги pre

[Reply](#)



•

Май 4, 2018 at 07:07

Сергей says:

Добрый день!

Есть какие-нибудь рекомендации и предпочтения по поводу использования конструктора и сеттера?

```
ObjectBox ob = new ObjectBox();
```

```
ob.setObject(obj);
```

или

```
ObjectBox ob = new ObjectBox(obj); //предварительно создав этот конструктор
```

[Reply](#)



o

Май 4, 2018 at 10:30

admin says:

Не могу выдать однозначные рекомендации 😊

Я не сторонник классов без конструктора по умолчанию — считаю, что он должен быть практически всегда. Исключение может быть в

случае классов, которые служат просто для хранения данных и есть желание делать их immutable (неизменными). Тогда есть смысл передать параметры сразу в конструкторе. Иногда можно сделать конструктор с параметрами для удобства — не делать два вызова — создание и потом установка через сеттер. Но т.к. это именно для удобства, то я не исключаю конструктор по умолчанию. Учитывая, что класс `ObjectBox` служебный и закрыт от посторонних, то его есть смысл разрабатывать так, чтобы было удобно использовать его именно для очереди. Наверно есть смысл добавить конструктор с параметром — именно для удобства.

[Reply](#)



Июнь 13, 2018 at 10:49

Дмитрий says:

3. Реализовать раздел Визуализация робота с помощью нашей очереди

Не понял третьего задания: Зачем и как это сделать?

Поставил в очередь и FIFO и LIFO отрезки пути Робота 10шт (объект `RobotLine`).

Забрал из очереди, обратно запомненные объекты, проверил, все верно.

Передал высвечивание линий по их координатам `Graphics g` в `paintComponent`.

Результат: окошко девственно пустое — отрезков нет.

Причина: что бы фигура высветилась GUI обращается к `paintComponent` минимум 2 раза — в первый раз из очереди ей все отрезки отдали, а во второй раз очередь уже пуста.

Можно конечно не забирать отрезки из очереди, а читать их оттуда `get` по каждому запросу `paintComponent` (а их 2,4, и более поступает от окна), но тогда какой смысл в очереди, если оттуда никто не уходит, это тот же список.

Можно конечно перегнать путь робота из очереди в массив или на тот же `ArrayList` и подсунуть этот массив (лист) в цикле под высветку, но чем это отличается от `Robot4`.

Что в `Robot4` ставить в очередь?

[Reply](#)



Июнь 13, 2018 at 11:32

admin says:

Если вы выполняли задание для раздела «Пример — очередь объектов» то вы должны были получить реализацию, которая позволяет пройти по всем элементам и не удалять их оттуда. По сути реализация списка.

[Reply](#)

Leave a reply


Comment

You may use these HTML tags and attributes: ` <abbr title=""> <acronym title=""> <blockquote cite=""> <cite> <code class="" title="" data-url=""> <del datetime=""> <i> <q cite=""> <s> <strike> <pre class="" title="" data-url=""> `

Имя *

E-mail *

Сайт

× 1 = два 

Add comment

Copyright © 2018 [Java Course](#)

Designed by [Blog templates](#), thanks to: [Free WordPress themes for photographers](#), [LizardThemes.com](#) and [Free WordPress real estate themes](#)

