

Professor Hans Noodles

41 уровень

22.07.2019   65921   18

# Структуры данных — стек и очередь

Статья из группы Java Developer  
43414 участников

Вы в группе

Привет!

Сегодня поговорим о таких важных для любого программиста вещах как **структуры данных**.



Википедия гласит:

**Структура данных** ([англ.](#) *data structure*) — программная единица, позволяющая хранить и обрабатывать множество однотипных и/или логически связанных данных в вычислительной технике.

Определение немного запутанное, но суть его ясна.

**Структура данных — это такое своеобразное хранилище, где мы держим данные для дальнейшего использования.**

В программировании существует огромное количество разнообразных структур данных.

**Очень часто при решении конкретной задачи самое главное — выбор наиболее подходящей для этого структуры данных.**

И со многими их них ты уже знаком! Например, с **массивами**. А также с **Мар** (которую обычно переводят как “словарь”, “карта”, или “ассоциативный массив”).

НАЧАТЬ ОБУЧЕНИЕ

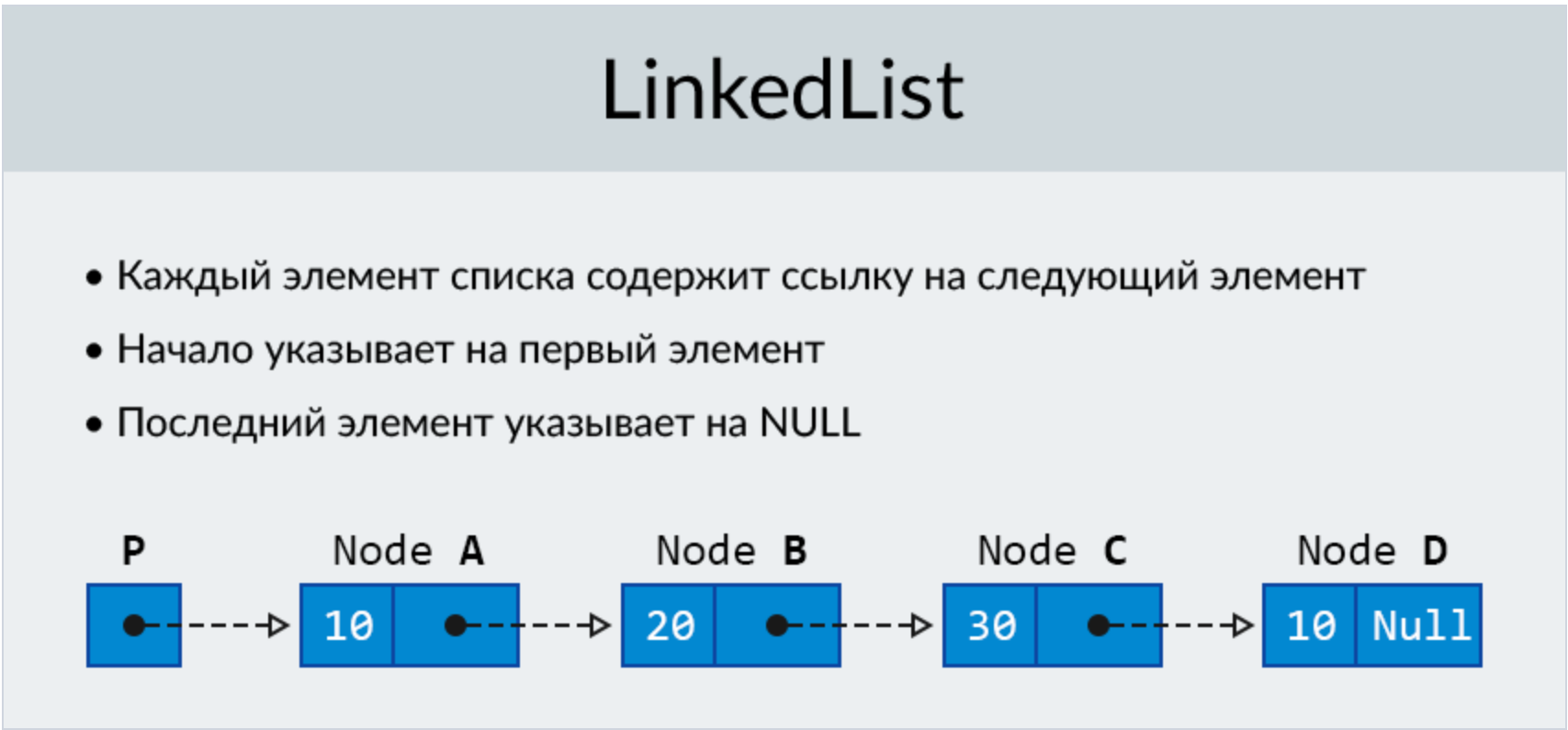
Это просто абстрактные “чертежи”, по которым каждый язык программирования создает свои собственные классы — реализации этой структуры.

Например, одна из самых известных структур данных — **связный список**. Ты можешь зайти в википедию, почитать о том как он устроен, какие у него есть достоинства и недостатки.

Возможно, тебе покажется знакомым его определение :)

“Свя́зный спи́сок — базовая динамическая структура данных в информатике, состоящая из узлов, каждый из которых содержит как собственно данные, так и одну или две ссылки («связки») на следующий и/или предыдущий узел списка”

Так ведь это же наш `LinkedList` !



Точно, так оно и есть :)

Структура данных “связный список” реализована в языке Java в классе `LinkedList`.

Но и в других языках связный список тоже реализован! В Python он называется “`llist`”, в Scala называется так же, как в Java — “`LinkedList`”.

Связный список — одна из базовых распространенных структур данных, поэтому ее реализацию ты найдешь в любом современном языке программирования.

То же самое с ассоциативным массивом. Вот его определение из Википедии:

**Ассоциативный массив** — абстрактный тип данных (интерфейс к хранилищу данных), позволяющий хранить пары вида «(ключ, значение)» и поддерживающий операции добавления пары, а также поиска и удаления пары по ключу.

Ничего не напоминает? :)

Точно, для нас, джавистов, ассоциативный массив — это интерфейс `Map`. Но эта структура данных реализована и в других языках! Например, программисты на C# знают его под названием “Dictionary”. А в языке Ruby он реализован в классе под названием “Hash”.

В общем, ты примерно понял в чем смысл: структура данных — это такая общая для всего программирования штука, которая реализуется по-своему в каждом конкретном языке.

Сегодня мы изучим две такие структуры и посмотрим, как они реализованы в Java — стек и очередь.

Стек — это известная структура данных.

Она очень проста и довольно много предметов из нашей повседневной жизни “реализованы” как стек.

Представь себе простую ситуацию: ты живешь в гостинице, и в течение дня тебе поступали деловые письма. Поскольку тебя в это время не было в номере, служащий гостиницы просто складывал приходящие письма на твой стол.

Сначала он положил на стол первое письмо.

Потом пришло второе, и он положил его **поверх** первого.

Третье пришедшее письмо он положил поверх второго, а четвертое — поверх третьего.



А теперь, ответь на простой вопрос: какое письмо ты прочтешь **первым**, когда придешь в номер и увидишь стопку на столе?

Правильно, ты прочитаешь **верхнее** письмо. То есть то, которое пришло **последним по времени**.

Именно так и работает стек. Такой принцип работа называется **LIFO — "last in — first out"** (“последним пришел — первым вышел”).

Для чего может пригодиться стек?

Например, ты создаешь на Java какую-то карточную игру. Колода карт лежит на столе. Отыгранные карты отправляются в сброс.

Ты можешь реализовать и колоду, и сброс используя два стека.

Игроки берут себе карты с верха колоды — тот же принцип, что и с письмами.

Когда игроки отправляют карты в сброс, новые карты ложатся поверх старых.

Вот как будет выглядеть первый набросок нашей игры, реализованный на основе стека:

```
1 public class Card {
2
3     public Card(String name) {
4         this.name = name;
5     }
6 }
```

[НАЧАТЬ ОБУЧЕНИЕ](#)

```
9      public String getName() {
10          return name;
11      }
12
13      public void setName(String name) {
14          this.name = name;
15      }
16
17      @Override
18      public String toString() {
19          return "Card{" +
20              "name='" + name + '\'' +
21              '}';
22      }
23  }
24
25  import java.util.Stack;
26
27  public class SimpleCardGame {
28
29      // колода
30      private Stack<Card> deck;
31
32      // сброс
33      private Stack<Card> graveyard;
34
35      public Card getCardFromDeck() {
36          return deck.pop();
37      }
38
39      public void discard(Card card) {
40          graveyard.push(card);
41      }
42
43      public Card lookTopCard() {
44
45          return deck.peek();
46      }
47
48      // ..геттеры, сеттеры и т.д.
49  }
```

Как мы и сказали ранее, у нас есть два стека: колода и сброс. Структура данных “стек” реализована в Java в классе `java.util.Stack`.

В нашей карточной игре есть 3 метода, описывающие действия игроков:

- взять карту из колоды (метод `getCardFromDeck()` );
- сбросить карту (метод `discard()` );
- посмотреть верхнюю карту(метод `lookTopCard()` ). Допустим, это будет бонусная механика “Разведка“, которая позволит игроку узнать, какая карта следующей попадет в игру.

Внутри наших методов вызываются методы класса `Stack`:

- `push()` — добавляет элемент на верх стека. Когда мы отправляем карту в сброс, она ложится поверх сброшенных ранее карт;



- `peek()` — возвращает верхний элемент стека, но не удаляет его из стека

Давай посмотрим, как будет работать наша игра:

```
1  import java.util.Stack;
2
3  public class Main3 {
4
5      public static void main(String[] args) {
6
7          // создаем колоду и добавляем в нее карты
8          Stack<Card> deck = new Stack<>();
9          deck.push(new Card("Рагнарос"));
10         deck.push(new Card("Пират Глазастик"));
11         deck.push(new Card("Сильвана Ветрокрылая"));
12         deck.push(new Card("Миллхаус Манашторм"));
13         deck.push(new Card("Эдвин ван Клифф"));
14
15         // создаем сброс
16         Stack<Card> graveyard = new Stack<>();
17
18         // начинаем игру
19         SimpleCardGame game = new SimpleCardGame();
20         game.setDeck(deck);
21         game.setGraveyard(graveyard);
22
23         // первый игрок берет 3 карты из колоды
24         Card card1 = game.getCardFromDeck();
25         Card card2 = game.getCardFromDeck();
26         Card card3 = game.getCardFromDeck();
27
28         System.out.println("Какие карты достались первому игроку?");
29         System.out.println(card1);
30         System.out.println(card2);
31         System.out.println(card3);
32
33         // первый игрок отправляет в сброс 3 своих карты
34         game.discard(card1);
35         game.discard(card2);
36         game.discard(card3);
37
38         System.out.println("Какие карты находятся в сбросе?");
39         System.out.println(game.getGraveyard().pop());
40         System.out.println(game.getGraveyard().pop());
41         System.out.println(game.getGraveyard().pop());
42     }
43 }
```

Итак, мы добавили в нашу колоду пять карт. Первый игрок взял 3 из них.

Какие же карты ему достались?

Вывод в консоль:

```
Card{name='Эдвин ван Клифф'}
Card{name='Миллхаус Манашторм'}
Card{name='Сильвана Ветрокрылая'}
```

Обрати внимание, в каком порядке карты были выведены в консоль.

Карта “Эдвин ван Клифф” в колоду попала последней (пятой по счету), и именно ее игрок взял первой. “Миххлаус” попал в колоду предпоследним, и его игрок взял вторым. “Сильвана” попала в колоду третьей с конца, и досталась игроку третьей.

Далее игрок сбрасывает карты. Сначала он сбрасывает Эдвина, потом Миллхауса, потом Сильвану.

После чего мы поочередно выводим в консоль карты, которые лежат у нас в сбросе:

**Вывод в консоль:**

```
Card{name='Сильвана Ветрокрылая'}
Card{name='Миллхаус Манашторм'}
Card{name='Эдвин ван Клифф'}
```

И снова мы видим как работает стек! Сброс в нашей игре тоже является стеком (как и колода).

“Эдвин ван Клифф” был сброшен первым. Вторым был сброшен “Миллхаус Манашторм” — и лег поверх Эдвина в сбросе. Далее была сброшена Сильвана — и эта карта легла уже поверх Миллхауса.

Как видишь, ничего сложного в работе стека нет. Тем не менее, знать эту структуру данных необходимо — о ней довольно часто спрашивают на собеседованиях, а на ее основе нередко строятся более сложных структуры данных.

## Очередь (Queue)

Очередь (или, по-английски, “Queue”) — еще одна распространенная структура данных.

Также как стек она реализована во многих языках программирования, в том числе и в Java.

В чем же отличие очереди от стека? Ее **очередь основана не на LIFO, а на другом принципе — FIFO (“first in — first out”, “первым вошел — первым вышел”)**.

Его легко понять, взяв для примера...да хотя бы обычную, настоящую очередь из реальной жизни! Например, очередь в магазин.



Если в очереди стоит пять человек, **первым** в магазин попадет тот, кто встал в очередь **первым**. Если еще один человек (помимо пяти в очереди) захочет что-то купить и встанет в очередь, то в магазин он попадает **последним**, то есть шестым.

При работе с очередью новые элементы добавляются в конец, а если ты хочешь получить элемент, он будет взят из начала.

Это основной принцип ее работы, который нужно запомнить



Принцип работы очереди очень легко понять интуитивно, поскольку она часто встречается в реальной жизни.

Стоит отдельно заметить, что в Java очередь представлена не классом, а интерфейсом — `Queue`.

Но вместе с тем, очередь в Java — это интерфейс, у которого есть очень много реализаций.

Если мы заглянем в документацию Oracle, то увидим, что от очереди наследуются 4 разных интерфейса, и крайне внушительный список классов:

All Known Subinterfaces

`BlockingDeque<E>`, `BlockingQueue<E>`, `Deque<E>`, `TransferQueue<E>`

All Known Implementing Classes

`AbstractQueue`, `ArrayBlockingQueue`, `ArrayDeque`

НАЧАТЬ ОБУЧЕНИЕ

LinkedList, LinkedList, LinkedList, LinkedList

PriorityBlockingQueue, PriorityQueue, SynchronousQueue

Какой большой список!

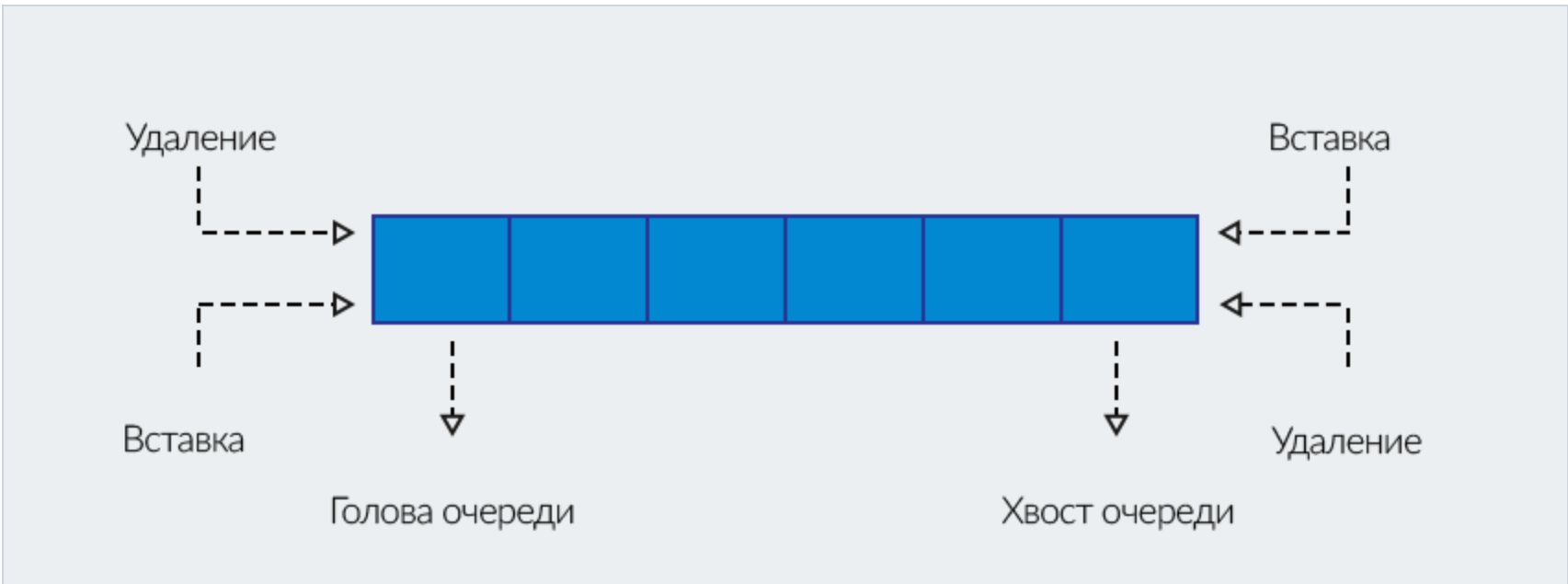
Но, конечно, тебе не нужно заучивать все эти классы и интерфейсы сейчас — голова может лопнуть :)

Мы рассмотрим лишь пару самых важных и интересных моментов.

Во-первых, обратим внимание на один из четырех “суб-интерфейсов” очереди — **Deque**. Чем он примечателен?

**Deque** — это двусторонняя очередь.

Она расширяет функционал обычной очереди, позволяя добавлять элементы на оба края (в начало и конец очереди) и забирать элементы с обоих краев очереди.



Двусторонняя очередь широко используется в разработке.

Обрати внимание на список классов-очередей, которые мы привели выше. Список довольно велик, но нет ли там чего-то знакомого для нас?

LinkedList, LinkedList, LinkedList, LinkedList

Ха, так здесь же наш старый знакомый — **LinkedList**!

То есть, он имплементирует интерфейс **Queue**? Но как он может быть очередью? Ведь **LinkedList** — это связный список!

Верно, но это никак не мешает ему быть очередью :) Вот список всех интерфейсов, которые он реализует:

All Implemented Interfaces:

Serializable, Cloneable, Iterable<E>, Collection<E>, Deque<E>, List<E>, Queue<E>

Как видишь, **LinkedList** реализует интерфейс **Deque** — двустороннюю очередь.



Благодаря этому мы можем получать элементы и из начала, и из конца `LinkedList`. А еще — добавлять элементы и в начало, и в конец.

Вот какие методы достались `LinkedList` от интерфейса `Deque`:

- `peekFirst()` — возвращает (но не удаляет из очереди) первый элемент.
- `peekLast()` — возвращает (но не удаляет из очереди) последний элемент.
- `pollFirst()` — возвращает первый элемент из очереди и удаляет его.
- `pollLast()` — возвращает последний элемент из очереди и удаляет его.
- `addFirst()` — добавляет новый элемент в начало очереди.
- `addLast()` — добавляет элемент в конец очереди.

Как видишь, `LinkedList` в полной мере реализует функционал двусторонней очереди!

И если такой функционал понадобится в программе,будешь знать, что его можно использовать :)

А наша сегодняшняя лекция подходит к концу.

### Научитесь программировать с нуля с JavaRush:

1200 задач, автопроверка решения и стиля кода

НАЧАТЬ ОБУЧЕНИЕ

Напоследок я дам тебе пару ссылок для дополнительного чтения.

Во-первых, обрати внимание на [статью, посвященную PriorityQueue](#) — “приоритетной очереди”.

Это одна из самых интересных и полезных реализаций Queue. Например, если в очереди в твой магазин стоят 50 человек, и 7 из них являются VIP-клиентами, `PriorityQueue` позволит тебе обслужить их в первую очередь! Очень полезная штука, согласен? :)

Во-вторых, не будет лишним еще раз упомянуть [книгу Роберта Лафоре “Структуры данных и алгоритмы на Java”](#). Во время чтения книги ты не только изучишь много структур данных (включая стек и очередь), но и самостоятельно реализуешь многие из них! Представь, например, что в Java не было бы класса Stack. Что бы ты делал, если бы тебе понадобилась такая структура данных для твоей программы? Конечно, пришлось бы написать ее самому. При чтении [книги Лафоре](#) ты часто именно этим и будешь заниматься. Благодаря этому твое понимание структур данных будет намного шире, чем при простом изучении теории :)

С теорией мы на сегодня закончили, но теория без практики — ничто!

Задачи сами себя не решат, так что самое время взяться за них! :)

−

+130

+

Комментарии (18)

популярные

новые

старые

JavaCoder

Введите текст комментария

НАЧАТЬ ОБУЧЕНИЕ

Магсумова Диана

Уровень 27, Россия

14 августа, 12:39

...

Прекрасная статья! )))) Благодарю))))

Ответить

Евгений Т.

Уровень 35, Москва, Россия

19 мая, 07:40

...

Ошибка в тексте "И со многими ИХ них ".

Ответить

Жора Нет

Уровень 39, енакиево, Украина

30 апреля, 22:10

...

Из лекций ничего нового не почерпнул, кроме определения сложности алгоритмов.

Все эти Стэки и Очереди приходилось самому находить и изучать при решении некоторых задач.

Так вот в чем суть этих лекций?

Я понимаю, если бы углублялись в изучении этих структур.

А так получается я уже о них знаю больше, чем мне только что поверхностно рассказали.

Ответить

Bulkin

Уровень 40, Нефтекамск, Россия

10 ноября 2021, 15:09

...

откуда столько времени найти такие книги читать?

Ответить

LuneFox

инженер по сопровождению в BIFIT

EXPERT

18 февраля, 18:10

...

программируешь в итоге машину времени и возвращаешься в тот день, когда начинал читать все эти книги

а если серьезно, то да, нужно скорее готовиться к работе, а не тратить месяцы на чтение войны и мира, мб во время работы уже почитать понемногу?

Ответить

Anonymous #2491313

Java Developer в Росатом

25 апреля 2021, 08:41

...

Почему доставание элемента из очереди называется pull, а не pop?

Ответить

LuneFox

инженер по сопровождению в BIFIT

EXPERT

18 февраля, 18:10

...

poll, а не pull

Ответить

Арман

Уровень 37, Самара, Россия

14 апреля 2021, 12:25

...

а на ее основе нередко строятся более сложныХ структуры данных.

Ответить

Максим Павлюк

Уровень 0

5 апреля 2021, 19:00

...

Я не понял откуда взялись эти классы и что они делают в примере про стек:

```
game.setDeck(deck);
game.setGraveyard(graveyard);
```

Ответить

Edil Kalmamatov

Уровень 35

12 октября 2021, 23:34

...

стандартные сеттеры для частных полей deck и graveyard. взялись из описания класса SimpleCardGame :

1

// ..геттеры, сеттеры и т.д.

Ответить

Иван


Уровень 41, Москва

28 декабря 2020, 19:53

...

Дайте несколько задач на очереди!

Ответить

 Виктор

веду учебный тг-канал в t.me/Javangelion

EXPERT

6 ноября 2020, 16:27

...

~~For the Alliance!~~

Спасибо за статью, достаточно доступно, понятно и с хорошими примерами.

Можно на эту тему ещё почитать следующие статьи:

[Стек-трейс Java.](#)

[Stack Trace и с чем его едят.](#)

Soros

Уровень 39, Харьков, Украина

20 апреля 2020, 20:05

...

Вы серьёзно?  
Т.е. конец, куда добавляют - в голове очереди, а начало, откуда забирают - в хвосте очереди?

Ответить

+14

Дмитрий Дмитрий

Уровень 35, Новосибирск, Россия

5 мая 2020, 11:15

...

тоже подогнался=))

Ответить

+1

Anthon Petrow

QA Manual Engineer в inDriver

9 июня 2020, 16:34

...

данное высказывание меня тоже загнал в непонятку

Ответить

0

Pig Man

Главная свинья в Свинарнике

15 февраля 2021, 20:12

...

Я не понял, что тебе в схеме не нравится. Слева - голова очереди (оттуда забирается элемент == удаляется), справа - конец очереди (туда добавляется новый элемент). Элемент из головы взяли - очередь продвинулась.  
  
Ты в магазине на кассе куда в очередь встаешь? В конец или в начало? В конец. Кого первым обслужат? Того, кто в начале. Что тебя повергло в недоумение?

Ответить

0

LuneFox

инженер по сопровождению в BIFIT

EXPERT

18 февраля, 18:14

...

Думаю, человек имел в виду, что постановка в очередь должна производиться в хвост, а не в голову. Ровно как и удаление элементов должно происходить из головы.

Ответить

0

Показать еще комментарии

ОБУЧЕНИЕ

- Курсы программирования
- Курс Java
- Помощь по задачам
- Подписки
- Задачи-игры

СООБЩЕСТВО

- Пользователи
- Статьи
- Форум
- Чат
- Истории успеха
- Активности

КОМПАНИЯ

- О нас
- Контакты
- Отзывы
- FAQ
- Поддержка



JavaRush — это интерактивный онлайн-курс по изучению Java-программирования с нуля. Он содержит 1200 практических задач с проверкой решения в один клик, необходимый минимум теории по основам Java и мотивирующие фишки, которые помогут пройти курс до конца: игры, опросы, интересные проекты и статьи об эффективном обучении и карьере Java-девелопера.

ПОДПИСЫВАЙТЕСЬ

ЯЗЫК ИНТЕРФЕЙСА

Русский

НАЧАТЬ ОБУЧЕНИЕ

