

Algorithms and data structures

lecture #9. Stack and Queue

Mentor: Miftakhov Il'yas

lecture #9. Stack and Queue

- Stack in Java

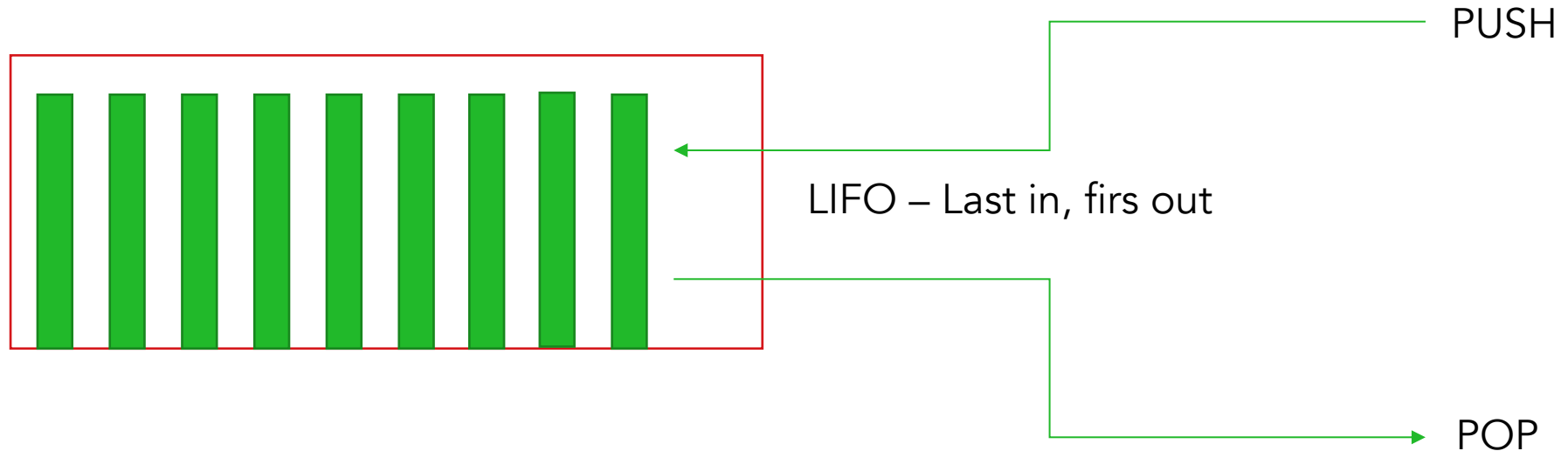
- Stack data structure
- Stack Class in Java
- Methods in Stack Class
- Задача getMin()
- Выводы

- Queue in Java

- Что это?
- Creating Queue Objects
- PriorityQueue
- Methods in PQ
- Выводы

Stack data structure

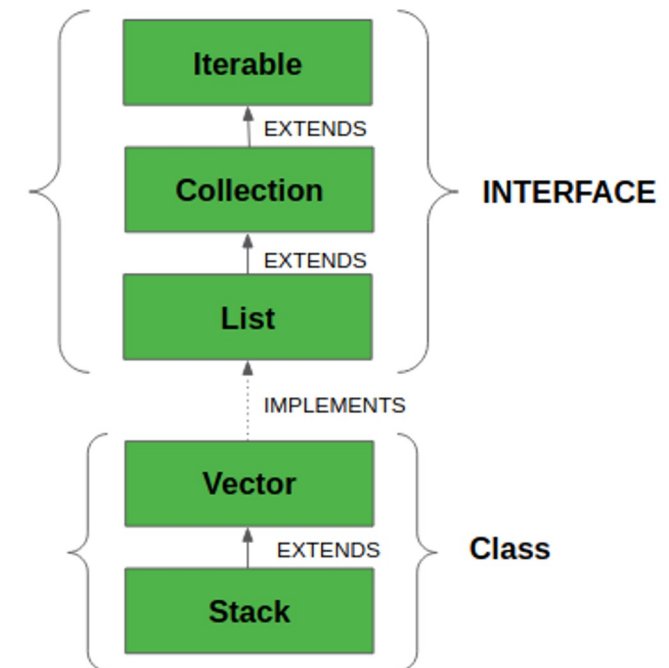
- Стек — это линейная структура данных, которая следует определенному порядку выполнения операций.
- Порядок LIFO (последним пришел, первым ушел).



Stack Class in Java

- Java Collection предоставляет класс Stack, который моделирует и реализует структуру данных Stack.
- Класс основан на основном принципе «последний пришел – первый ушел».
- В дополнение к основным операциям push и pop класс предоставляет еще три функции: empty, search и peek.

Класс поддерживает один конструктор по умолчанию Stack() , который используется для создания пустого стека .



Доступные методы

- **empty()** – Он возвращает true, если на вершине стека ничего нет. В противном случае возвращает false.
- **peek()** – Возвращает элемент с вершины стека, но **не удаляет** его.
- **pop()** – Удаляет и возвращает верхний элемент стека.
 - Исключение «EmptyStackException» Исключение возникает, если мы вызываем pop(), когда стек пуст.
- **push(Object element)** – Помещает элемент на вершину стека.
- **search(Object element)** - Определяет, существует ли объект в стеке.
 - Если элемент найден, возвращает **позицию элемента** с вершины стека. В противном случае он возвращает -1.

Задача getMin()

- Реализовать структуру данных SpecialStack, которая поддерживает все операции со стеком, такие как push(), pop(), isEmpty(), ... и дополнительную операцию **getMin()**, которая должна возвращать **минимальный элемент** из SpecialStack.
- Все эти операции SpecialStack должны быть $O(1)$. Пространство $O(n)$
- Чтобы реализовать SpecialStack, вы должны использовать только стандартную структуру данных Stack и никакие другие структуры данных, такие как массивы, списки, файлы и т.п.

Рассмотрим следующий SpecialStack

```
16 --> ВЕРХ
15
29
19
18
```

Когда вызывается getMin(), он должен возвращать 15, который является минимальным элементом в текущем стеке.

Если мы вытолкнем два раза из стека, стек станет

```
29 --> ВЕРХ
19
18
```

Когда вызывается getMin(), он должен вернуть 18 который является минимумом в текущем стеке.

Выводы

- Все случаи в которых нам нужны данные, которые пришли последними
 - Например реализация карточной игры
- Класс Stack в Java является устаревшим классом и наследуется от Vector в Java.
- Это потокобезопасный класс и, следовательно, требует накладных расходов, когда нам не нужна потокобезопасность.
- Для реализации стека рекомендуется использовать ArrayDeque, так как он более эффективен в однопоточной среде.

Queue in Java

- **Интерфейс** Queue присутствует в пакете `java.util` и расширяет интерфейс `Collection`, используемый для хранения элементов, которые должны быть обработаны, в порядке **FIFO** (первым пришел — первым ушел).
- Это упорядоченный список объектов, использование которого ограничено вставкой элементов в **конец списка** и удалением элементов из **начала списка** (FIFO(- First In First Out
- Очередь нуждается в **конкретном** классе для объявления.
- Наиболее распространенными классами являются `PriorityQueue` и `LinkedList` в Java.
- Ни одна из этих реализаций не является потокобезопасной. `PriorityBlockingQueue` — одна из альтернативных реализаций, если требуется поточно-ориентированная реализация.

Создание объектов Queue

- Поскольку Queue является интерфейсом, объекты типа Queue не могут быть созданы.
- Нам **всегда** нужен класс, который расширяет этот список, чтобы создать объект.
- После введения Generics в Java 1.5, можно ограничить тип Queue, который может храниться в Queue.
- `Queue<Object> queue = new PriorityQueue<Object>();`

Operations on Queue Interface - PriorityQueue

- **Добавление элементов.**
- Чтобы добавить элемент в очередь, мы можем использовать метод **add()**.
- Порядок размещения **не сохраняется в PriorityQueue**.
- Элементы сохраняются в порядке **приоритета**, который по умолчанию является возрастающим.

Operations on Queue Interface - PriorityQueue

- **Удаление элементов.**
- Чтобы удалить элемент из очереди, мы можем использовать метод **remove()**.
- Если таких объектов несколько, **первое вхождение** объекта удаляется.
- Кроме того, метод **poll()** также используется для удаления головы и ее возврата.

Operations on Queue Interface - PriorityQueue

- **Итерация очереди.**
- Существует несколько способов итерации очереди.
- Самый известный способ — **преобразование очереди** в массив и обход с помощью цикла for.
- Очередь также имеет **встроенный итератор**, который можно использовать для перебора очереди.

Characteristics of a Queue: выводы

- Java Queue используется для вставки элементов в конец очереди и удаления из начала очереди. Он следует концепции FIFO.
- Java Queue поддерживает все методы интерфейса Collection, включая вставку, удаление и т. д.
- LinkedList , ArrayBlockingQueue и PriorityQueue — наиболее часто используемые реализации.
- Если над BlockingQueues выполняется какая-либо пустая операция, генерируется исключение NullPointerException.
- Все очереди, кроме Deques, поддерживают вставку и удаление в конце и в начале очереди соответственно. Deques поддерживают вставку и удаление элементов с обоих концов.