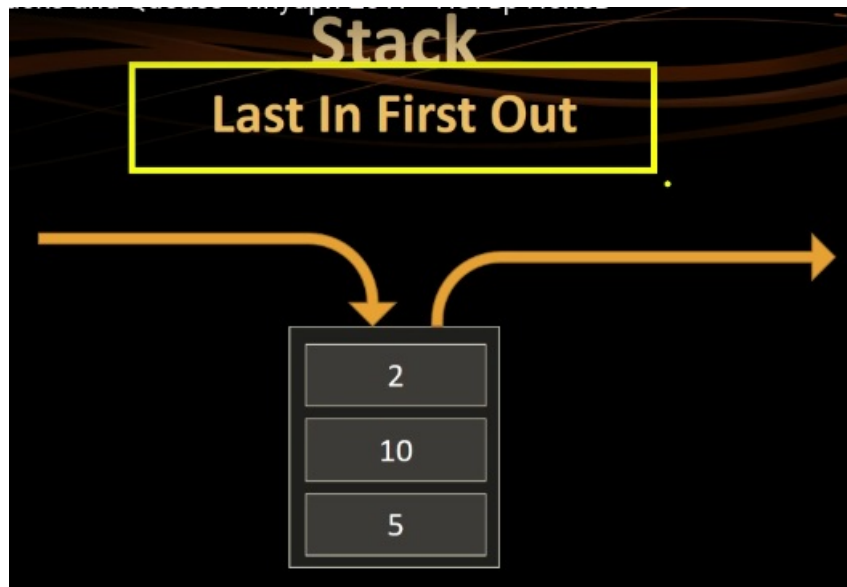


Стек. - Колекция от данни, която има поведение на last in, first out (тоест последния вкаран елемент, излиза първи). Показано е втората картинка за стека.

## 1. Stack<E> (LIFO – last in, first out)

- Stack Functionality - `push()`, `pop()`, `peek()`
- Stack Utilities - `toArray()`, `contains()` and `size()`



Стека поддържа няколко функционалности:

- `push()` - Слага елемент най-отгоре както е показано на следващата снимка.



- `pop()` - Изважда елемент от стека.



- `peek()` - Само виждаме елементът, който седи най-отгоре, но не го изваждаме от стека. Остава си там.



Как се създава Стек в Java ?

### ■ Creating a Stack

```
ArrayDeque<Integer> stack = new ArrayDeque<>();
```

По принцип в Java си има структура от данни, която се нарича стек и може да се използва, но е СИЛНО НЕПРЕПОРЪЧИТЕЛНО (Има проблеми с performance-a). Дори Oracle казват, че не трябва да се използва :)

Използва се кодът показан на следващата картинка.

### ■ Creating a Stack

```
ArrayDeque<Integer> stack = new ArrayDeque<>();
```

### ■ Adding elements at the top of the stack

```
stack.push(element);
```

Премахването на елементи отново става с pop() както е показано на следващата картинка.

### ■ Removing elements

```
Integer element = stack.pop();
```

peek() се използва по същия начин като pop -> stack.peek()

Трябва да се внимава с типовете данни. ArrayDeque не позволява да се създаде каквато и да е колекция с примитивен тип данни както ясно се вижда от следваща картинка. Всъщност всяка колекция, която съдържа <> не може да се инициализира с примитивен тип данни.

```
ArrayDeque<int> stack = new ArrayDeque<>(); - неправилен начин.
```

```
ArrayDeque<Character> stack = new ArrayDeque<>(); - правилен начин
```

Utility methods или помощни средства на стека.

```
ArrayDeque<Integer> stack = new ArrayDeque<>();

Integer size = stack.size();
boolean isEmpty = stack.isEmpty();
boolean exists = stack.contains(2);
Integer[] arr = stack.toArray();
```

stack.contains(2) търси по стойност, а не по индекс. Ако искаме индекс трябва да напишем нещо подобно

```
stack.toArray()[1];
```

В stack.toArray() се запазва последователността на елементите. В какъвто ред сме ги сложили в стека така и ще излязат под формата на масив.

Колекцията Collections има метод addAll, тоест Collections.addAll(), чрез който може да се добавят елементи от една колекция в друга. На следващата картинка е показано първо как се взема ред от козолата, който обаче веднага е разделен на масив наречен tokens. След това се създава колекцията, която ще съдържа тези знаци или всичко, което е разбито по един или повече празни интервали - .split("\\s+"); След това редът Collections.addAll(stack, tokens), добавя всички токени, в колекцията stack.

```
public class Demo {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String[] tokens = scanner.nextLine().split("\\s+");

        ArrayDeque<String> stack = new ArrayDeque<>();
        Collections.addAll(stack, tokens);
    }
}
```

---

Опашка - Линейна структура от данни, която много наподобява на стека, само че реда на елементите е различен. Тук е FIFO - first in, first out, докато при стека е LIFO - last in first out.

### 3. Queue<E> (FIFO – first in, first out)

- Queue Functionality – offer(), poll()
- Queue Functionality - toArray(), contains(), size()

# Queue

## First In First Out



Абстрактен тип данни е просто идеята.

ArrayDeque-а се използва и за стек и за опашка, като разликата е методите, които се използват.

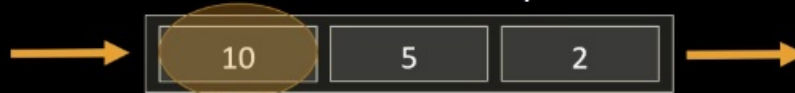
Добавянето, премахването и виждането на първия елемент в опашката е подобно на стека само че в обратен ред.

И трите метода са показани на следващата картинка.

## Queue – Abstract Data Type

- Queues provide the following functionality:

- Adding an element at the end of the queue



- Removing the first element from the queue



- Getting the first element of the queue without removing it



Създаване на опашка.

Опашката се създава по абсолютно същия начин както стека - чрез ArrayDeque. Разликата е единствено в методите, които се използват след това. При стека бяха push, pop, peek, а в опашката add/offer, remove/poll

- Creating a Queue

```
ArrayDeque<Integer> queue = new ArrayDeque<>();
```

Добавяне на елемент в опашката е ясно показано в следващият слайд:


- Creating a Queue

```
ArrayDeque<Integer> queue = new ArrayDeque<>();
```

- Adding elements at the end of the queue

```
queue.add(element);  
queue.offer(element);
```

- **add()** – throws exception if queue is full
- **offer()** – returns false if queue is full



`add()` добавя елемент в опашката, но хвърля `exception`(грешка) и се прекъсва изпълнението на програмата ако и се прехвърли размера.

`offer()` - прави същото като `add()`, само че не гърми(не хвърля грешка и не прекъсва изпълнението на програмата), а просто връща `false`.


Ако не ни интересува дали сме вкарали успешно елемента или не, може да използваме `offer`, в противен случай ползваме `add`. Всъщност и двата метода могат да се използват по един и същи начин, просто при `add` ще трябва да се `catch`-ва грешката, а при `offer` може просто да се проверява някаква булева стойност.

Махане на елементи от опашката е показано на следващия слайд:

- Removing elements

```
element = queue.remove();  
element = queue.poll();
```

- **remove()** – throws exception if queue is empty
- **poll()** – returns null if queue is empty



Разликата между двата метода е същата като при тези по-горе. `Remove` хвърля грешка и спира изпълнението на програмата в случай че няма елемент за премахване, а `poll()` връща `null` в същия случай.

Можем да проверим кой е първия елемент по начина показан на следваща картинка

- Check first element

```
element = queue.peek();
```

Има и друг метод, който проверява първия елемент и той е `peekFirst()` и прави абсолютно същото като `peek()`;

Как Java разбира дали програмистът е направил стек или опашка ?

И двете са еднакви, просто едното се движи отзад-напред(stack), а другото отпред-назад (queue)

ArrayDeck се нарича double ended queue, тоест опашка , която има два края. Ние сме избираме с кой от тях да работим. (Началото или с края). Можем да вмъкваме елементи и в началото и в края, както и ги премахваме по същия начин.