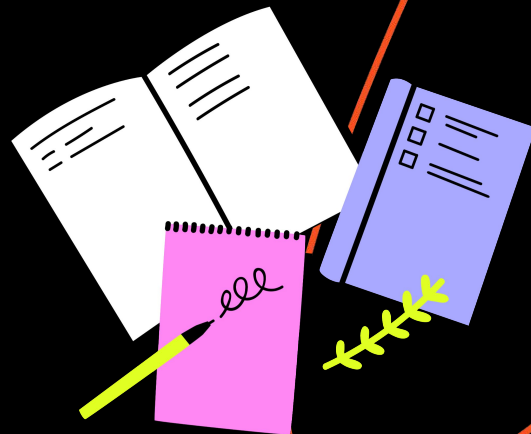




# Приоритетные коллекции

Хранение и обработка данных, часть I



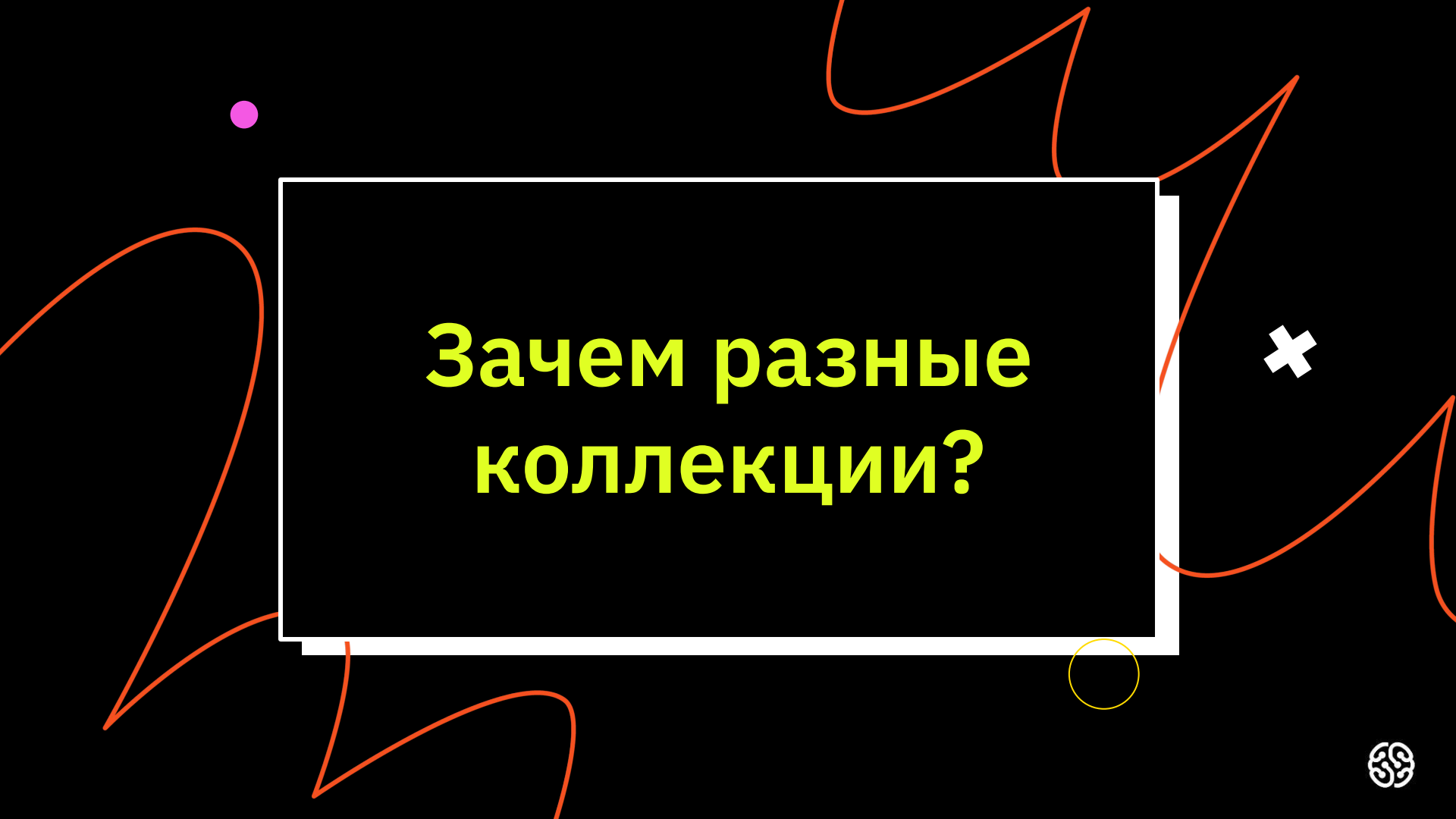
**У меня есть  
план**



# У меня есть план

1. Разобраться, зачем нужно столько коллекций и JCF
2. LinkedList и особенности работы с ним
3. Узнать о списках, как об абстрактной структуре
4. Проблемы работы со списками и массивами
5. Построение разных Queue и как в этом помогает JCF
6. Deque и логика использования этой коллекции
7. То, что мертво, умереть не может – Stack в контексте JCF





**Зачем разные  
коллекции?**



# Зачем разные коллекции?

1. Способ обработки данных
2. Решаемые задачи



# LinkedList



# LinkedList

Представляет собой двусвязный список.



# LinkedList

Представляет собой двусвязный список.

**Список** – гибкая структура данных, позволяющая легко менять свой размер. Элементы доступны для вставки или удаления в любой позиции.

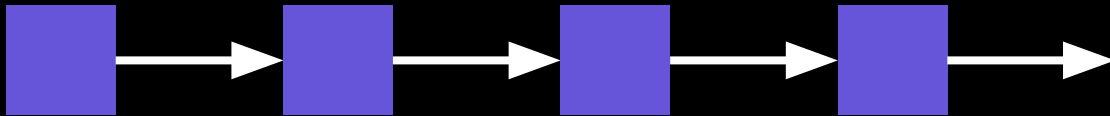




# LinkedList

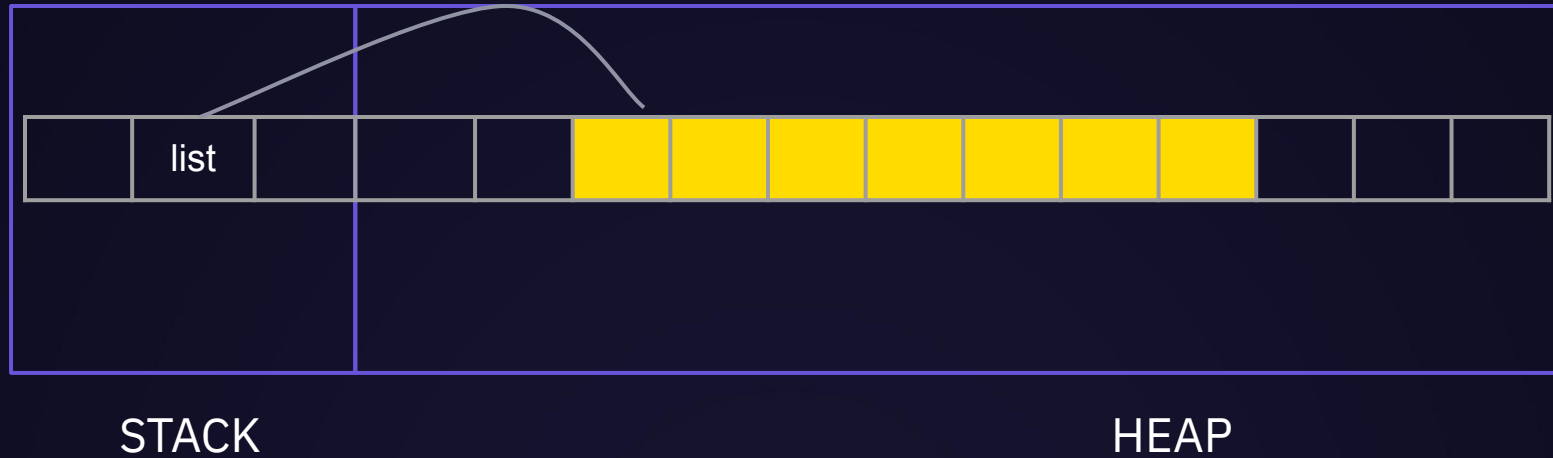
Представляет собой двусвязный список.

**Список** – гибкая структура данных, позволяющая легко менять свой размер. Элементы доступны для вставки или удаления в любой позиции.



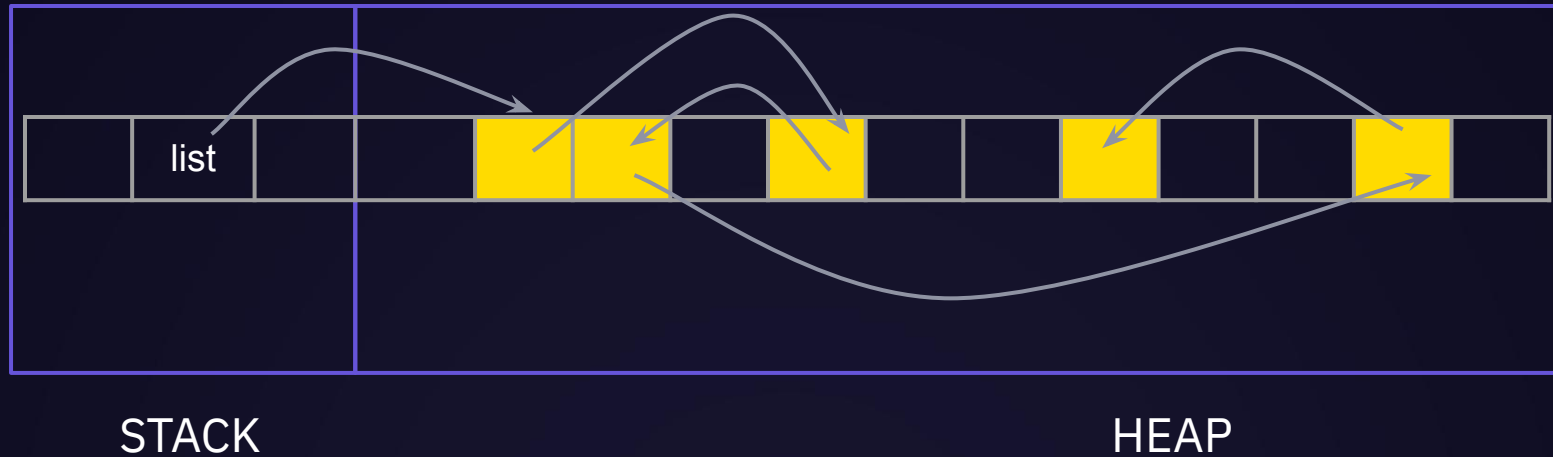
# LinkedList. Список

Массив



# LinkedList. Список

Связный список



# LinkedList. Список

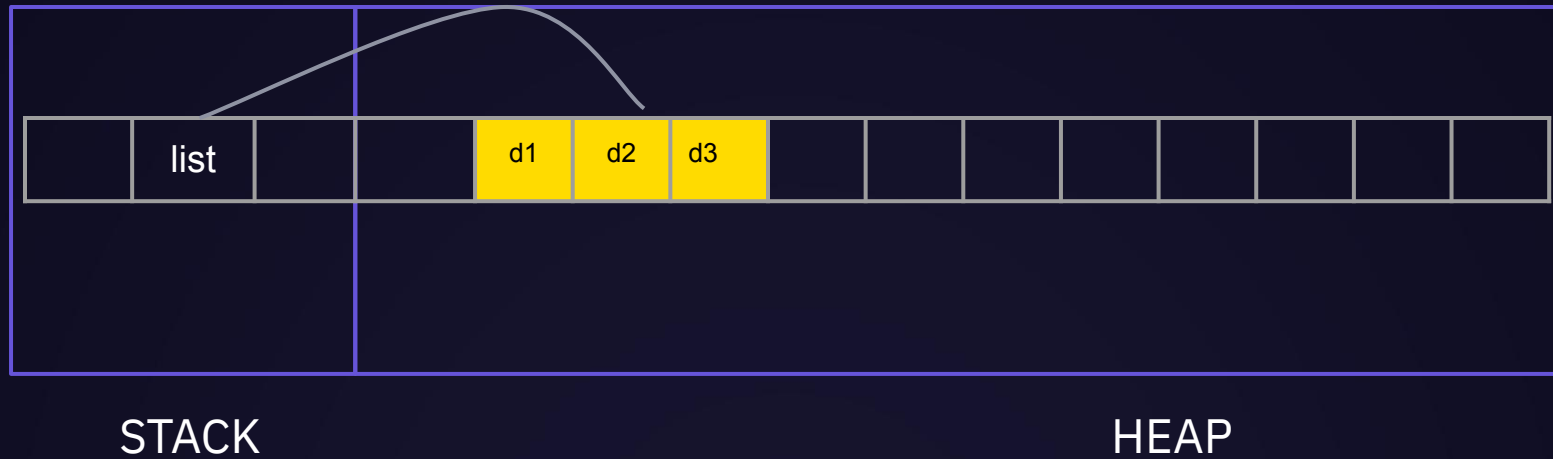
Чем такой способ организации лучше? Или хуже?

Проблема добавления в массив.



# LinkedList. Список

Массив



# LinkedList. Список

Массив



# LinkedList. Список

Массив



# LinkedList. Список

Массив





# LinkedList. Список

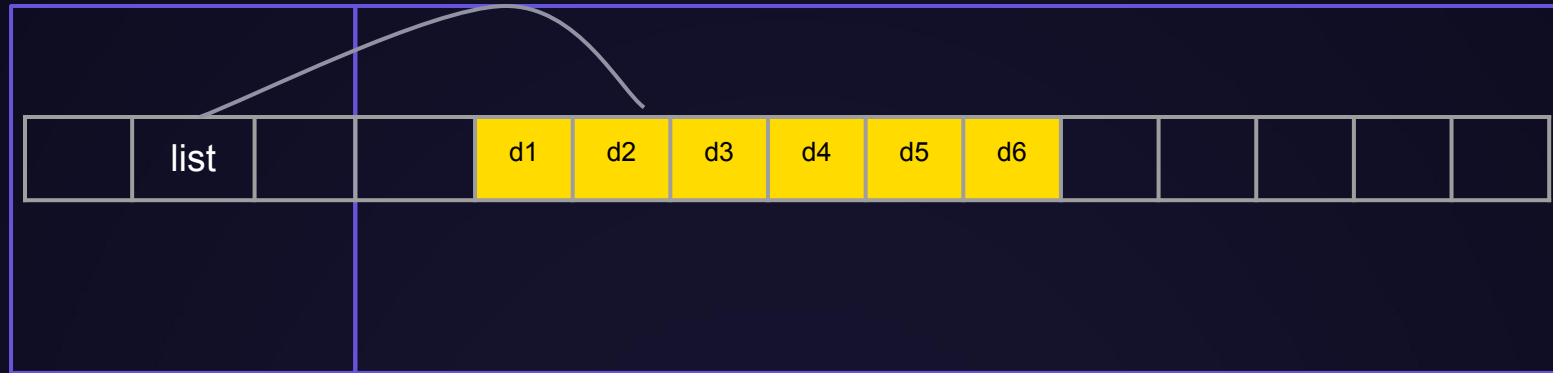
Чем такой способ организации лучше? ли хуже?

Проблема удаления из массива.



# LinkedList. Список

Массив



STACK

HEAP



# LinkedList. Список

Массив



# LinkedList. Список

Массив



# LinkedList. Список

Массив



# LinkedList. Список

Массив



# LinkedList. Список

Массив



STACK

HEAP



# LinkedList. Список

Массив





# LinkedList. Список

Чем такой способ организации лучше? Или хуже?

Попробуйте написать такой код – отсылка к предыдущим лекциям.

В качестве тренировки попробуйте добавить элемент не в хвост.



# LinkedList. Список

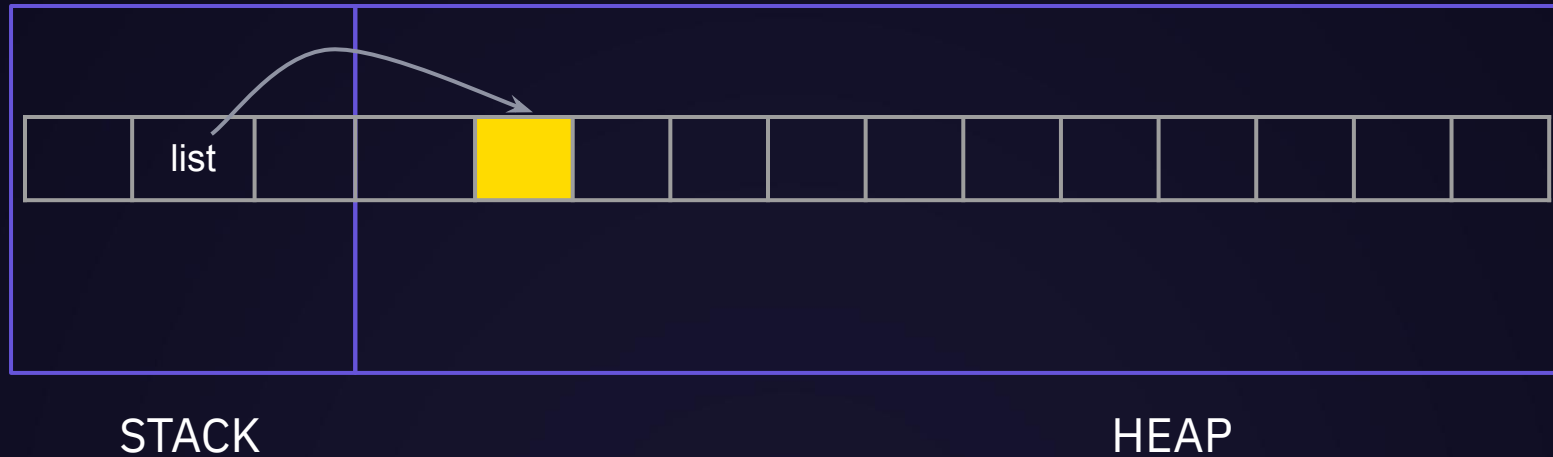
Чем такой способ организации лучше? Или хуже?

Добавление в конец списка.



# LinkedList. Список

Связный список



# LinkedList. Список

Связный список



# LinkedList. Список

Связный список



# LinkedList. Список

Связный список



# LinkedList. Список

Связный список



# LinkedList. Список

Связный список





# LinkedList. Список

Связный список



# LinkedList. Список

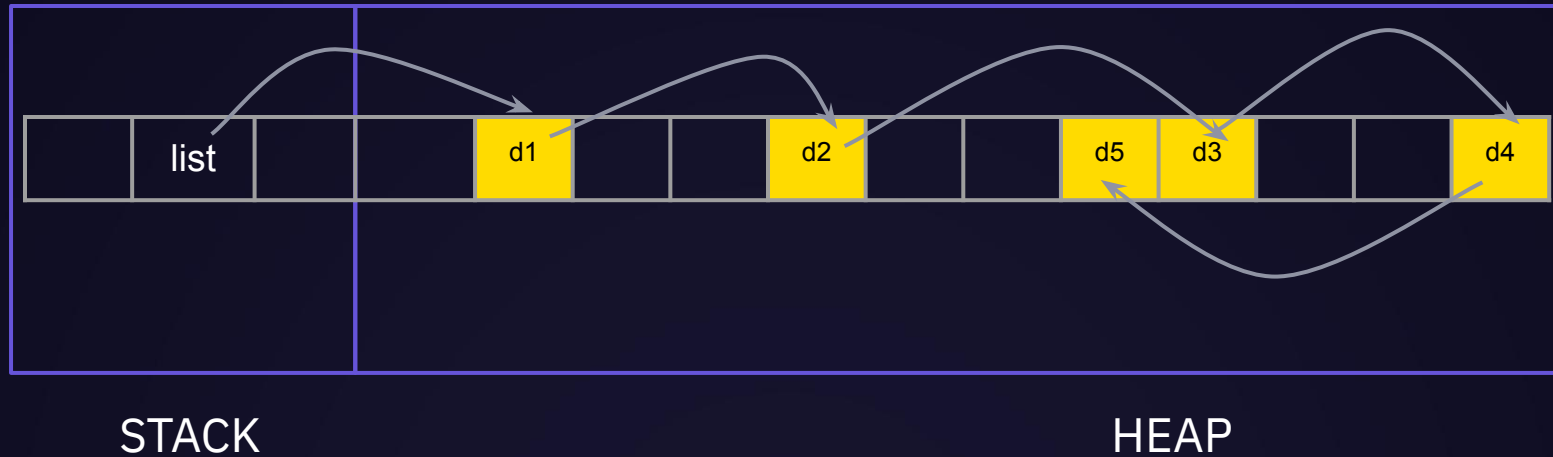
Чем такой способ организации лучше? Или хуже?

Добавление в середину списка.



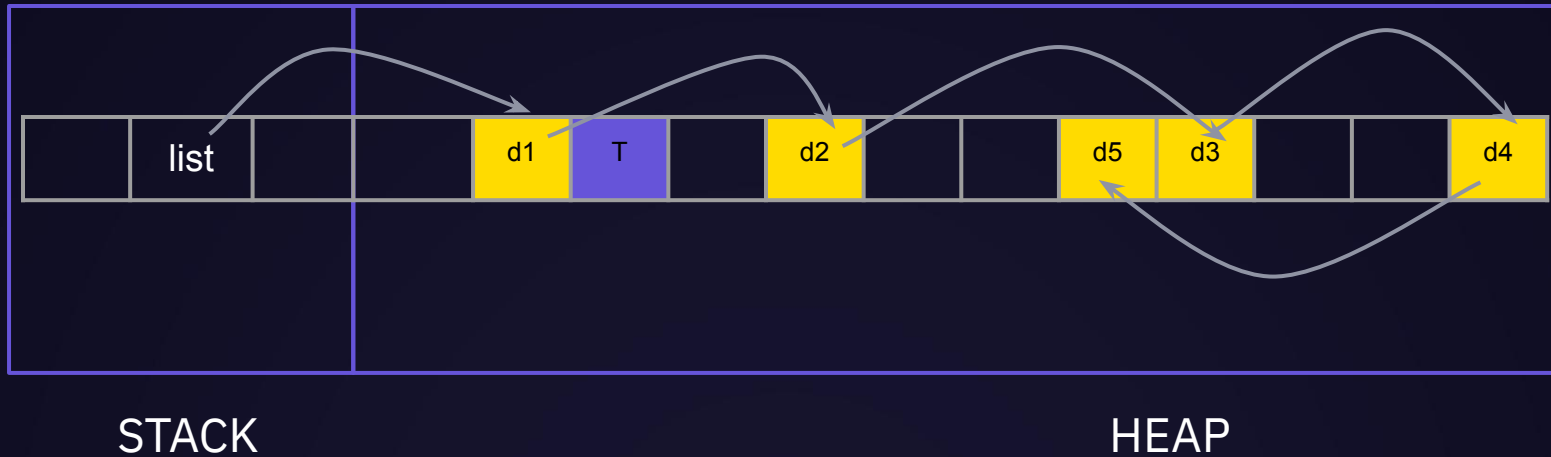
# LinkedList. Список

Связный список



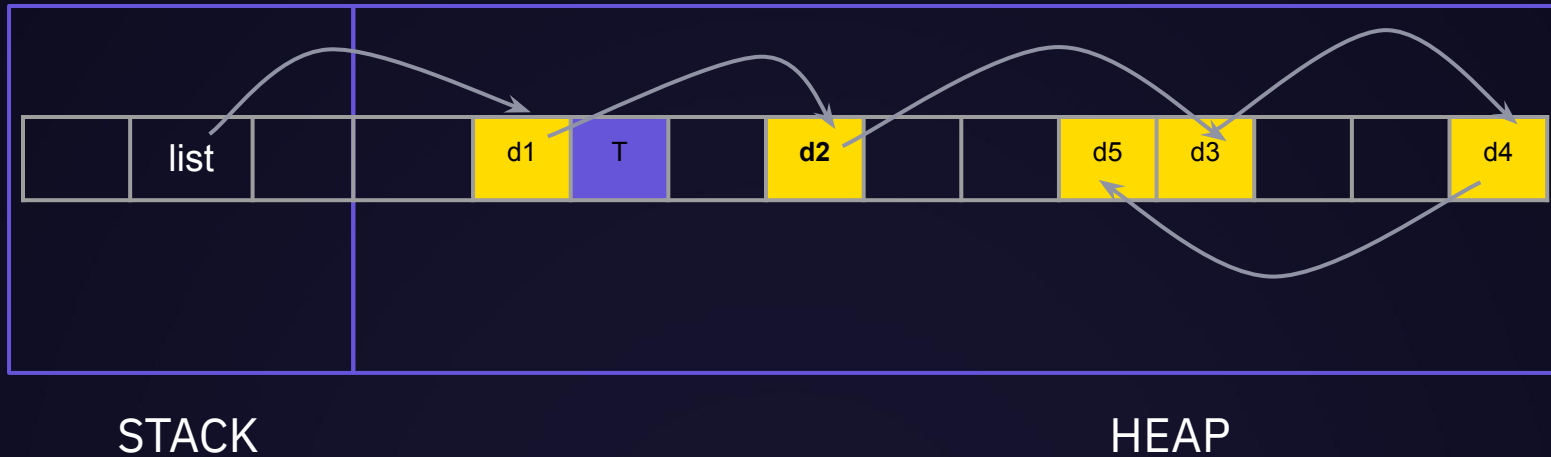
# LinkedList. Список

Связный список



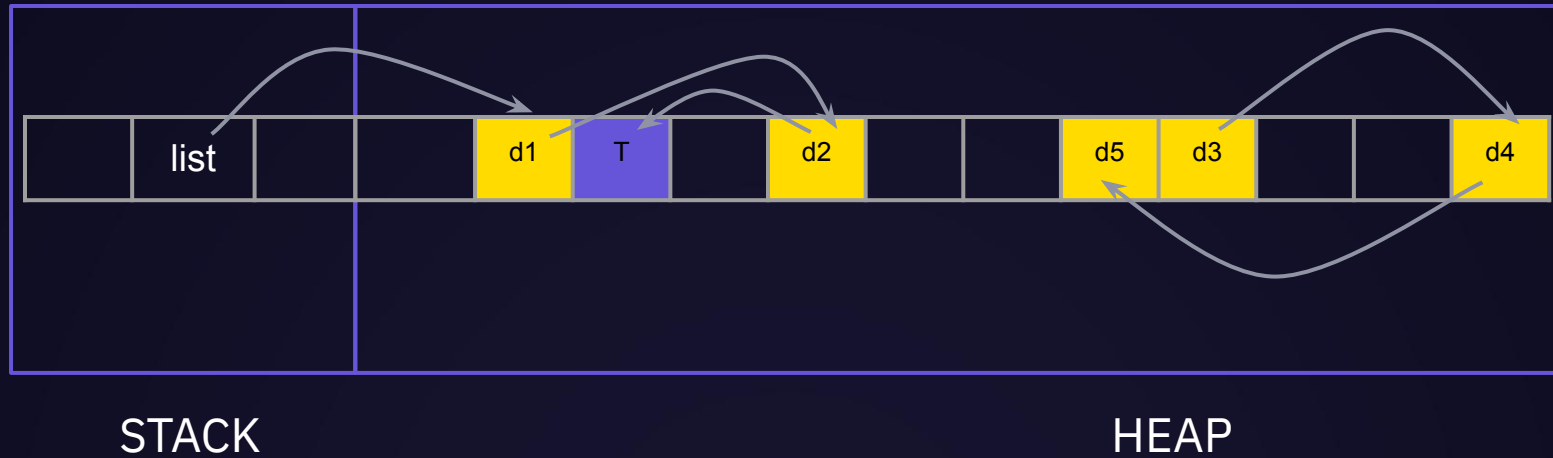
# LinkedList. Список

Связный список



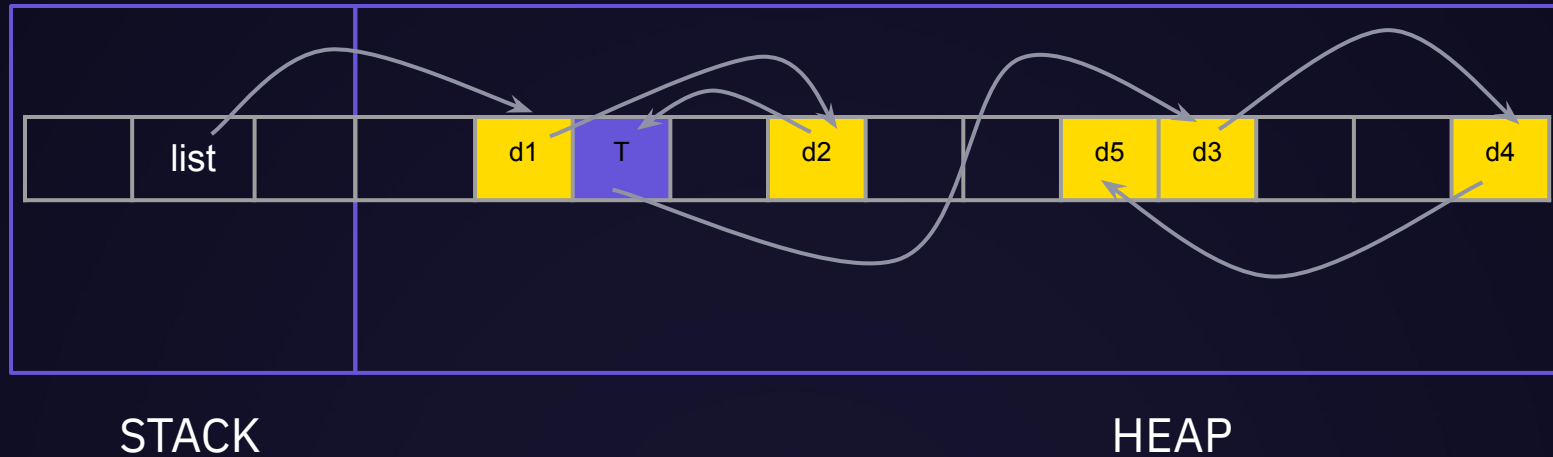
# LinkedList. Список

Связный список



# LinkedList. Список

Связный список



# LinkedList. Список

Чем такой способ организации лучше? Или хуже?

Удаление из списка списка.





# LinkedList. Список

Чем такой способ организации лучше? Или хуже?

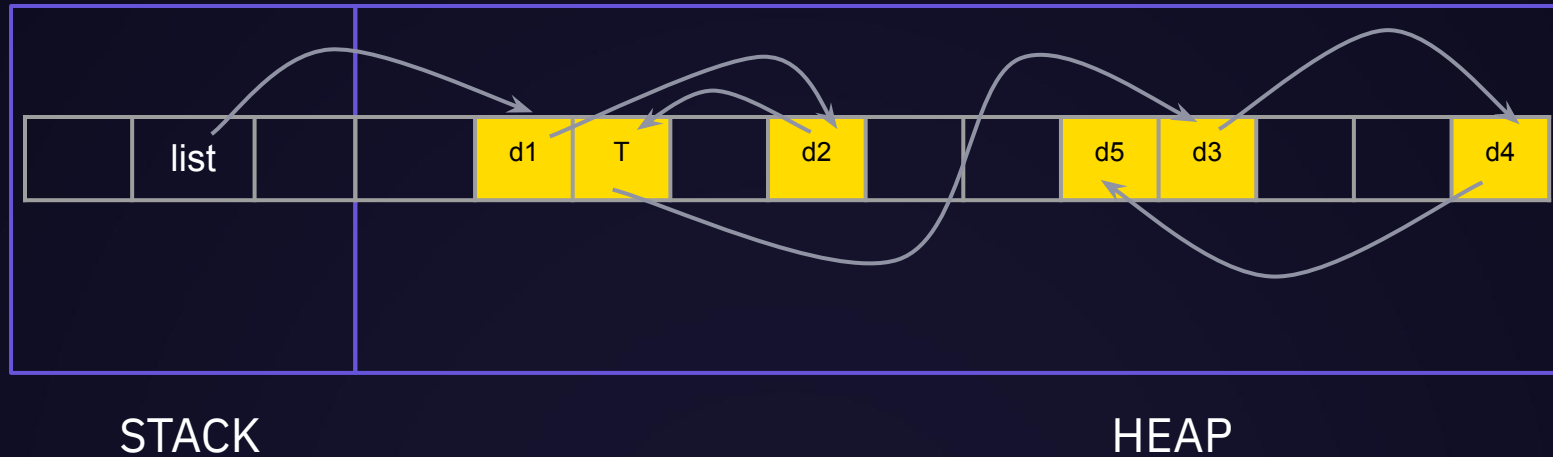
Удаление из списка списка.

Еще пример.



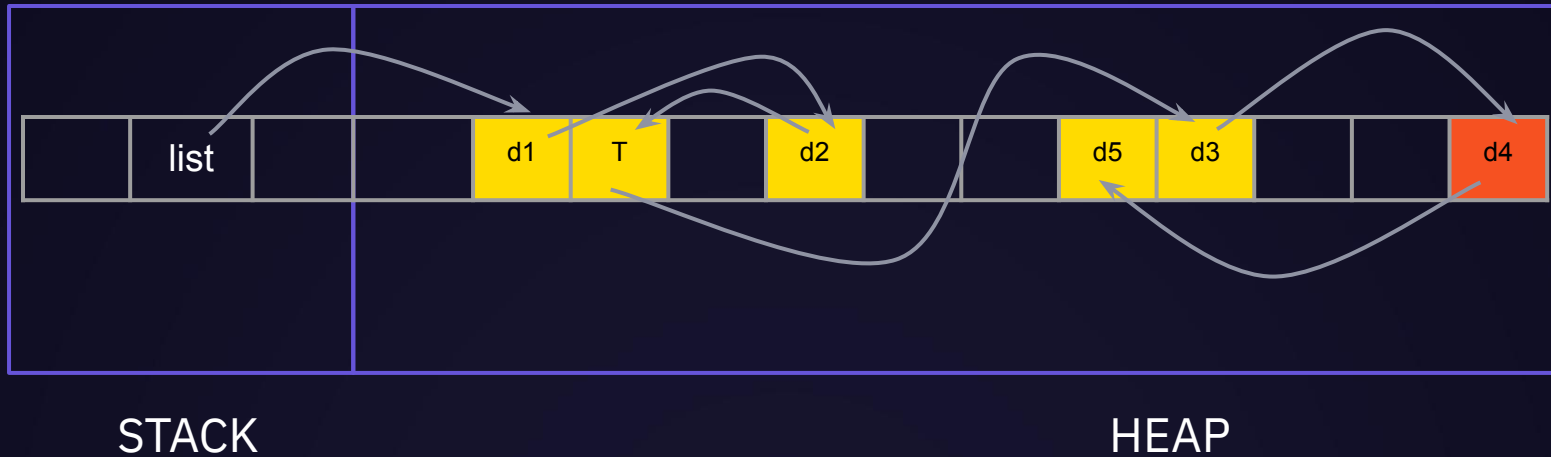
# LinkedList. Список

Связный список



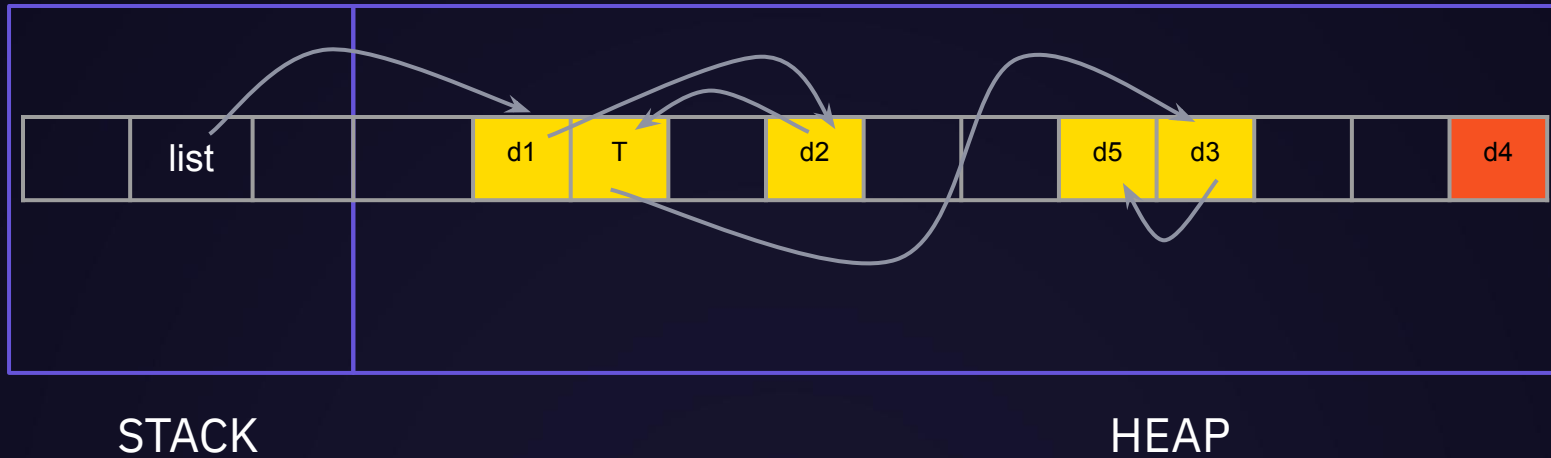
# LinkedList. Список

Связный список



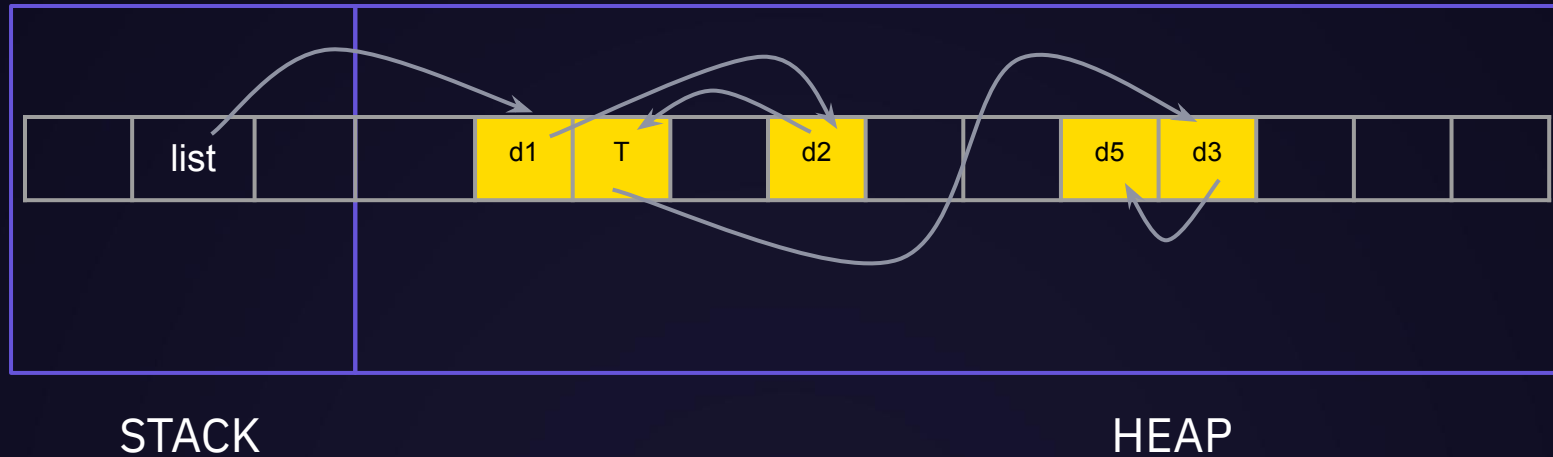
# LinkedList. Список

Связный список



# LinkedList. Список

Связный список



# LinkedList. Список

Чем такой способ организации лучше? Или хуже?

Что нам с этого?

Количество операций меньше => скорость доступа к данным выше.

Всегда ли? Доступ к элементу.



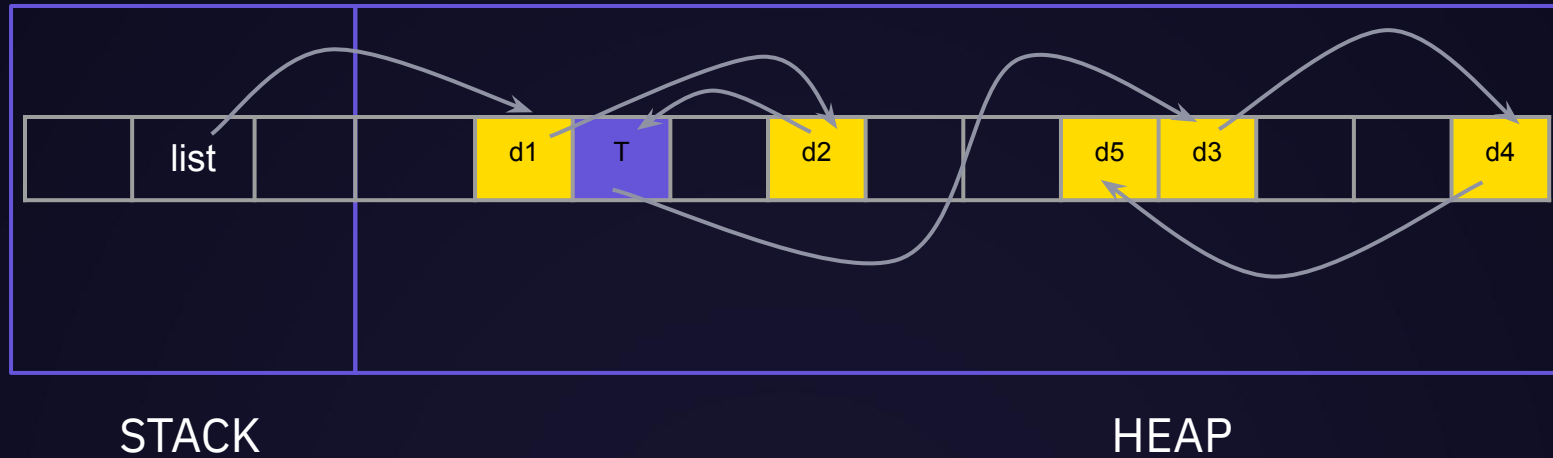
# LinkedList. Список

Массив



# LinkedList. Список

Связный список





# LinkedList. Список

Чем такой способ организации лучше? Или хуже?

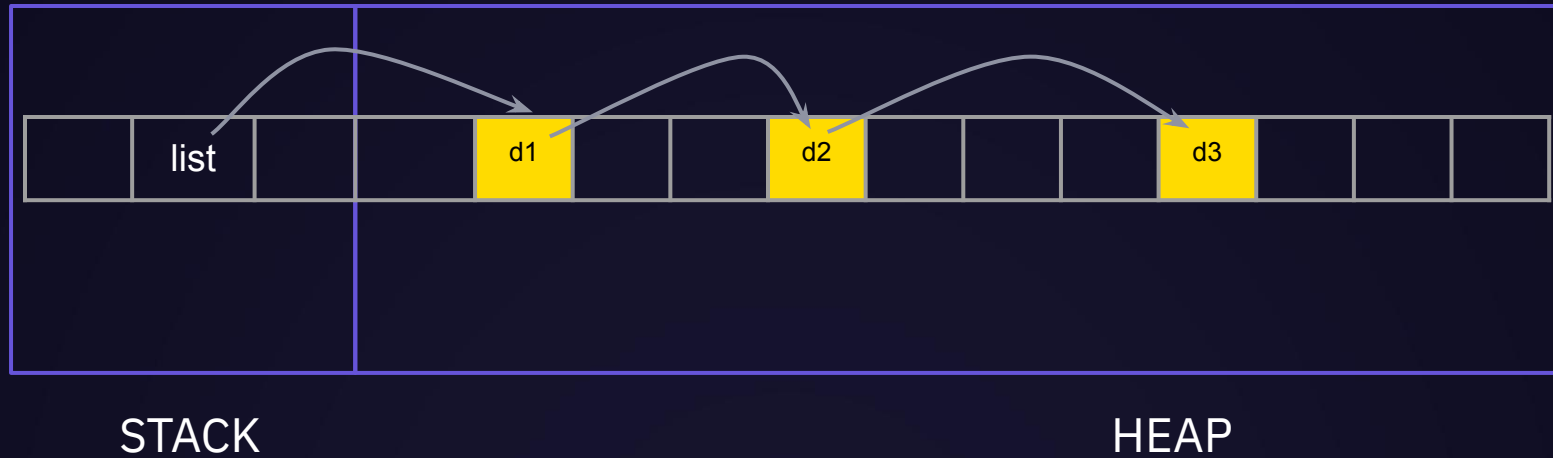
Односвязный список.

Двусвязный список.



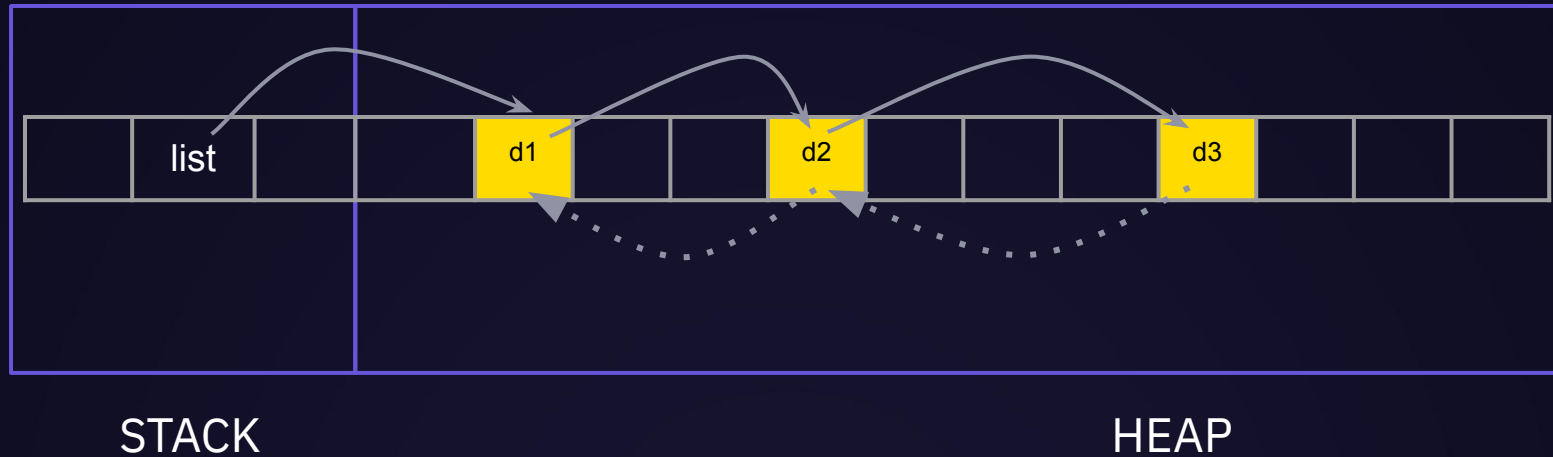
# LinkedList. Список

Связный список



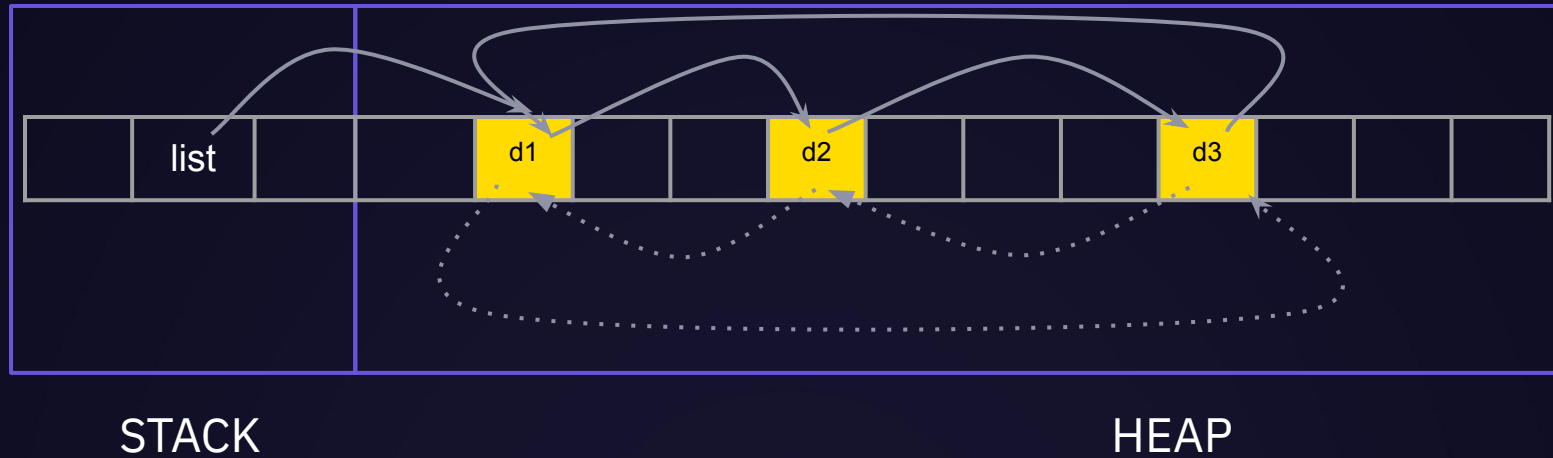
# LinkedList. Список

Связный список



# LinkedList. Список

Связный список



# LinkedList. Список

Мотивация.

LinkedList нужен для небольшого количества элементов, в которых операции добавления\удаления встречаются чаще операций чтения.



# LinkedList. Список

Чем такой способ организации лучше? Или хуже?

Потренируйтесь с операциями добавления\удаление в двусвязном и циклическом списках.



Object

Демонстрация





Queue





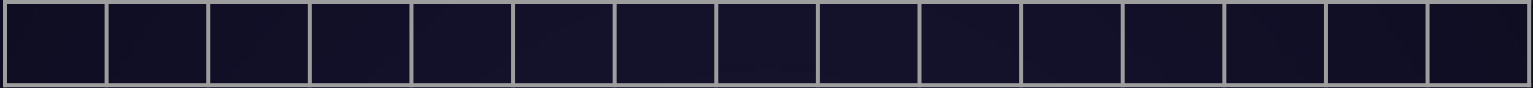
# Queue

Очередь в магазине



# Queue

```
Queue<Integer> queue = new LinkedList<Integer>();
```



# Queue

```
Queue<Integer> queue = new LinkedList<Integer>();  
queue.add(1);
```



# Queue

```
Queue<Integer> queue = new LinkedList<Integer>();  
queue.add(2);
```



# Queue

```
Queue<Integer> queue = new LinkedList<Integer>();  
queue.add(3);
```



# Queue

```
Queue<Integer> queue = new LinkedList<Integer>();  
queue.add(4);
```



# Queue

```
Queue<Integer> queue = new LinkedList<Integer>();  
int item = queue.remove();
```



# Queue

```
Queue<Integer> queue = new LinkedList<Integer>();  
int item = queue.remove();
```





# Queue

```
Queue<Integer> queue = new LinkedList<Integer>();  
queue.add(5);
```



# Queue

```
Queue<Integer> queue = new LinkedList<Integer>();  
queue.add(28);
```



# Queue

```
Queue<Integer> queue = new LinkedList<Integer>();  
int item = queue.remove();
```



# Queue

```
Queue<Integer> queue = new LinkedList<Integer>();  
int item = queue.remove();
```



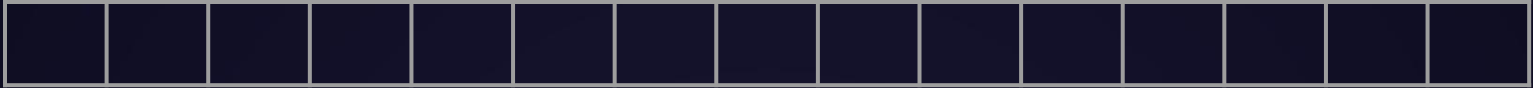
# Queue

```
Queue<Integer> queue = new LinkedList<Integer>();  
int item = queue.remove();
```



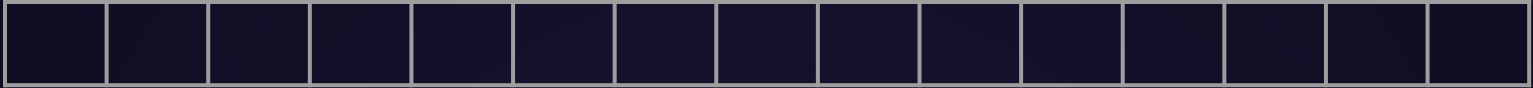
# Queue

```
Queue<Integer> queue = new LinkedList<Integer>();  
int item = queue.remove();
```



# Queue

```
Queue<Integer> queue = new LinkedList<Integer>();  
int item = queue.remove();
```



```
java.util.NoSuchElementException
```



Queue

Демонстрация





# Queue

```
import java.util.*;
public class Ex002_Queue {
    public static void main(String[] args) {
        Queue<Integer> queue = new LinkedList<Integer>();
        queue.add(1);
        queue.add(2);
        queue.add(3);
        queue.add(4);
        System.out.println(queue); // [1, 2, 3, 4]
        int item = queue.remove();
        System.out.println(queue); // [2, 3, 4]
        queue.offer(2809);
        item = queue.remove();
        System.out.println(queue); // [3, 4, 2809]
        item = queue.remove();
        System.out.println(queue); // [4, 2809]
        item = queue.poll();
        System.out.println(queue); // [2809]
    }
}
```



# Queue

```
import java.util.*;
public class Ex002_Queue {
    public static void main(String[] args) {
        Queue<Integer> queue = new LinkedList<Integer>();
        queue.add(1);
        int item = queue.remove();
        queue.offer(2809);
        item = queue.poll();
        System.out.println(queue);
    }
}
```



# Queue

```
import java.util.*;
public class Ex002_Queue {
    public static void main(String[] args) {
        Queue<Integer> queue = new LinkedList<Integer>();
        queue.add(1);
        int item = queue.remove();
        queue.offer(2809);
        item = queue.poll();
        System.out.println(queue);

        queue.element()
        queue.peek()
    }
}
```



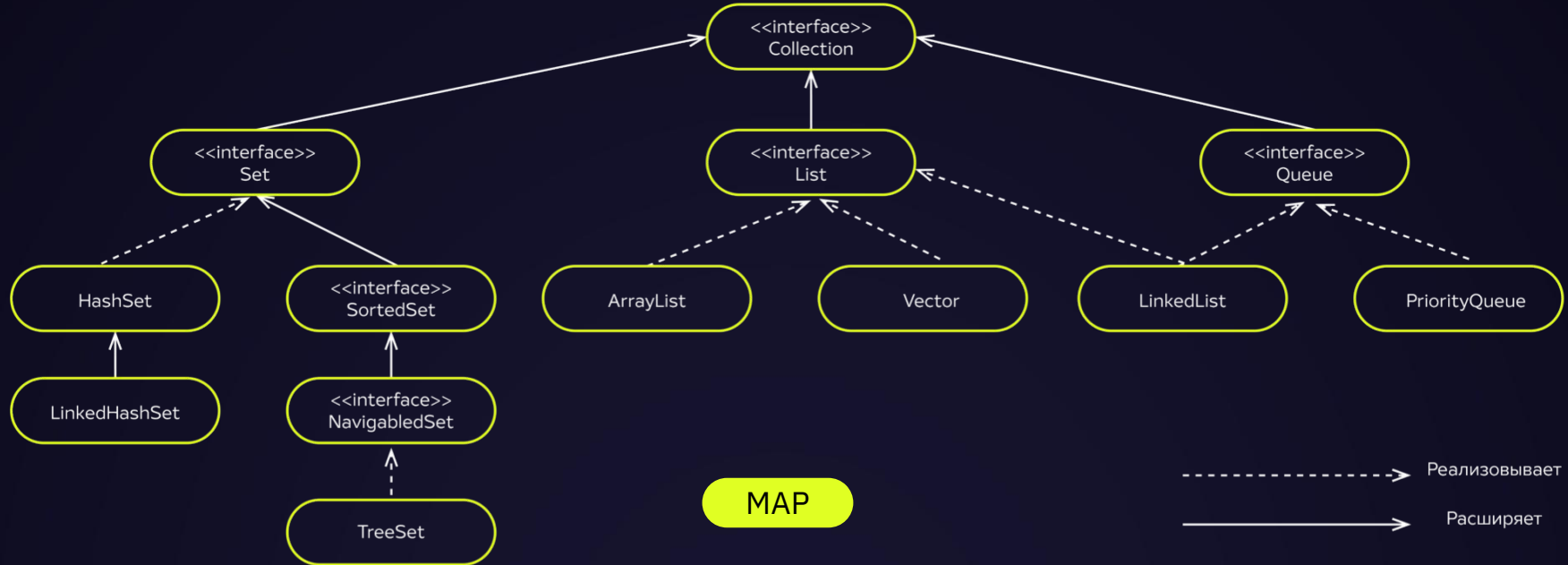
# Queue

```
import java.util.*;
public class Ex002_Queue {
    public static void main(String[] args) {
        Queue<Integer> queue = new LinkedList<Integer>();
        queue.add(1);
        int item = queue.remove();
        queue.offer(2809);
        item = queue.poll();
        System.out.println(queue);

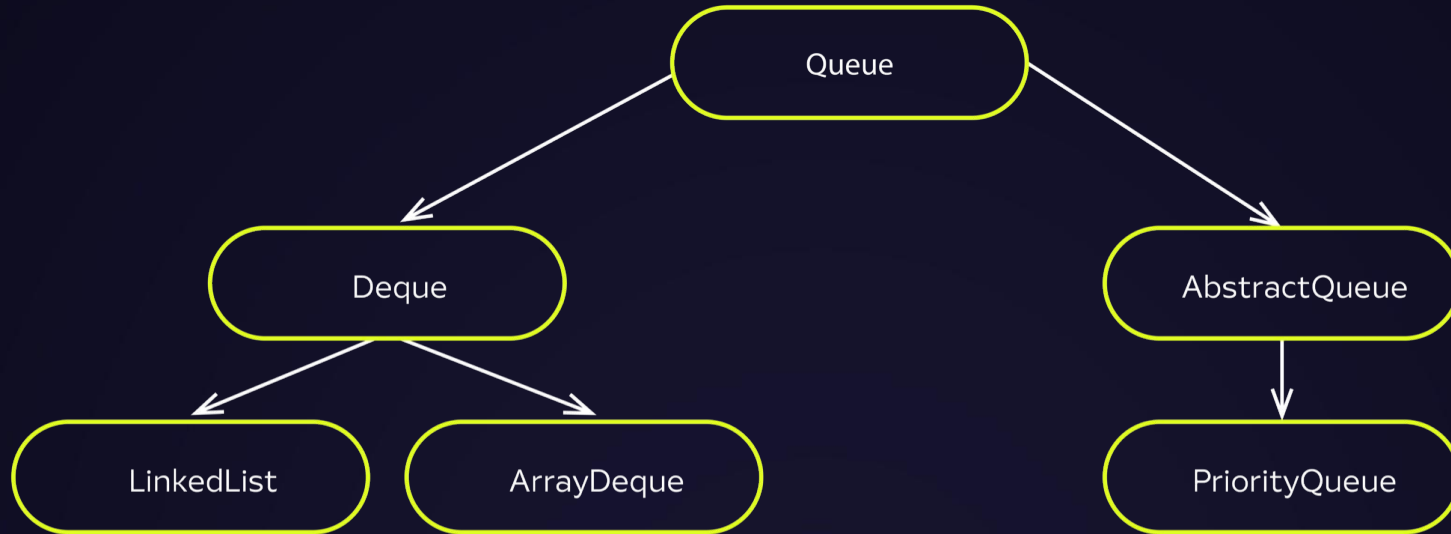
        queue.remove(2809); // зачем очередь??
        queue.element()
        queue.peek()
    }
}
```



# Иерархия коллекций



# Иерархия коллекций



# PriorityQueue



# PriorityQueue

Наивысший приоритет имеет «наименьший» элемент.

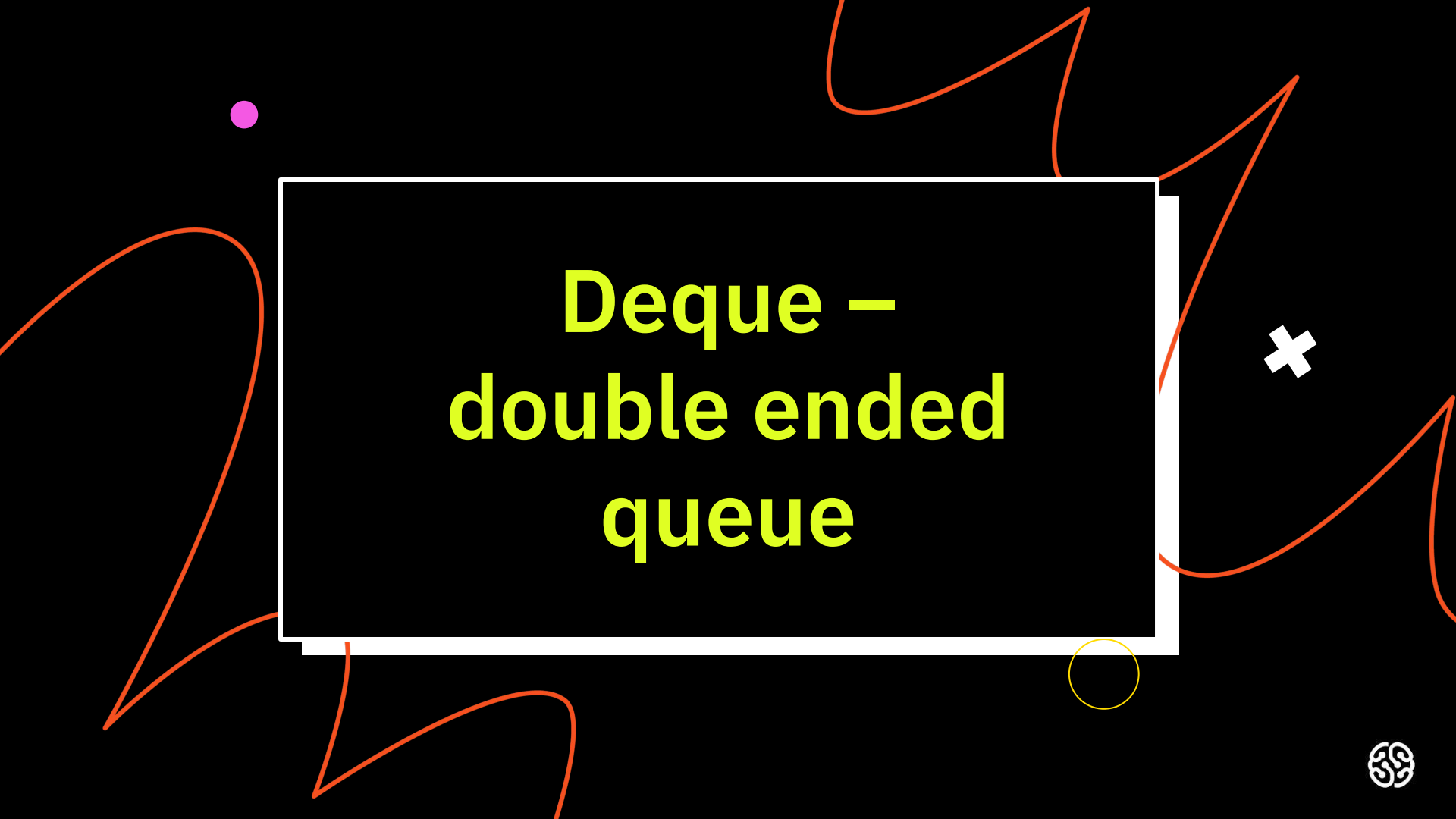




# PriorityQueue

## Демонстрация





# Deque – double ended queue



# Deque

Линейная коллекция, которая поддерживает вставку и удаление элементов на обоих концах. | [url](#)



# Deque

```
import java.util.*;
public class Ex003_Deque {
    public static void main(String[] args) {
        Deque<Integer> deque = new ArrayDeque<>();
        Deque<Integer> deque = new ArrayDeque<>();
        deque.addFirst(1);    deque.addLast(2);
        deque.removeLast();   deque.removeLast();
        deque.offerFirst(1);  deque.offerLast(2);
        deque.pollFirst();    deque.pollLast();
        deque.getFirst();     deque.getLast();
        deque.peekFirst();    deque.peekLast();
    }
}
```

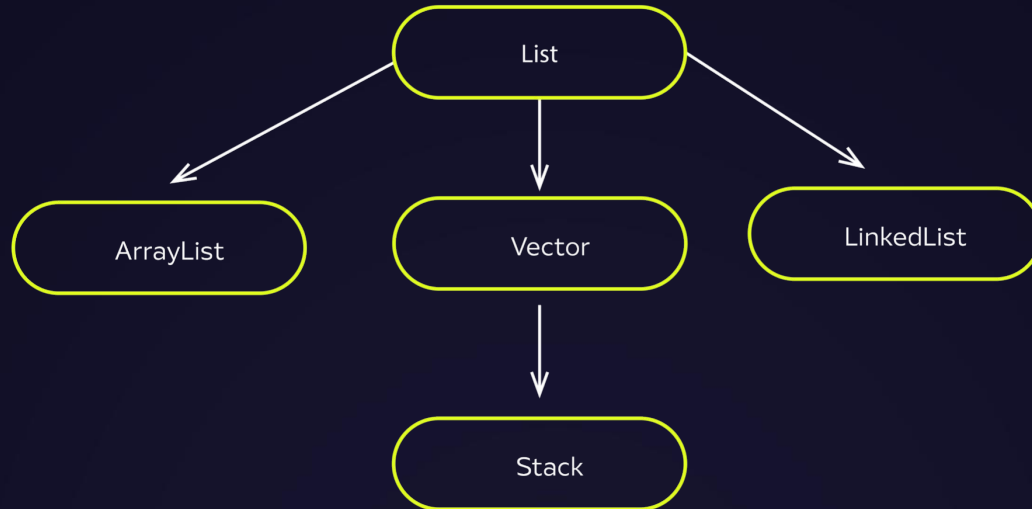




**Stack**



# Иерархия коллекций



# Stack

Stack представляет собой обработку данных по принципу LIFO.  
Расширяет Vector пятью операциями, которые позволяют рассматривать вектор как стек.

`push(E item)`

`pop()`



# Stack

```
import java.util.*;
public class Ex004_Stack {
    public static void main(String[] args) {
        Stack<Integer> stack = new Stack<>();
        stack.push(1);
        stack.push(12);
        stack.push(123);
        System.out.println(stack.pop()); // 123
        System.out.println(stack.pop()); // 12
        System.out.println(stack.pop()); // 1
    }
}
```





# Stack

Демонстрация  
Stack и Vector не рекомендованы  
к использованию



**Демка**



# Stack

Вычислить значение выражения в постфиксной форме записи



# Stack

Вычислить значение выражения в постфиксной форме записи

$1 + 2 * 3$        $(1 + 2) * 3$



# Stack

Вычислить значение выражения в постфиксной форме записи

$1 + 2 * 3$        $(1 + 2) * 3$   
 $1\ 2\ 3\ *\ +$



# Stack

Вычислить значение выражения в постфиксной форме записи

$1 + 2 * 3$	$(1 + 2) * 3$
$1\ 2\ 3\ *\ +$	$1\ 2\ +\ 3\ *$



# Stack

Вычислить значение выражения в постфиксной форме записи

$1 + 2 * 3$	$(1 + 2) * 3$
$1\ 2\ 3\ *\ +$	$1\ 2\ +\ 3\ *$

1 2 3 \* +

СТЕК



# Stack

Вычислить значение выражения в постфиксной форме записи

$1 + 2 * 3$	$(1 + 2) * 3$
$1\ 2\ 3\ *\ +$	$1\ 2\ +\ 3\ *$

1 2 3 \* +

СТЕК





# Stack

Вычислить значение выражения в постфиксной форме записи

$1 + 2 * 3$	$(1 + 2) * 3$
$1\ 2\ 3\ *\ +$	$1\ 2\ +\ 3\ *$

2 3 \* +

СТЕК  
1



# Stack

Вычислить значение выражения в постфиксной форме записи

$1 + 2 * 3$	$(1 + 2) * 3$
$1\ 2\ 3\ *\ +$	$1\ 2\ +\ 3\ *$

2 3 \* +

СТЕК  
1



# Stack

Вычислить значение выражения в постфиксной форме записи

$1 + 2 * 3$	$(1 + 2) * 3$
$1\ 2\ 3\ *\ +$	$1\ 2\ +\ 3\ *$

$3\ *\ +$

СТЕК

2

1



# Stack

Вычислить значение выражения в постфиксной форме записи

$1 + 2 * 3$	$(1 + 2) * 3$
$1\ 2\ 3\ *\ +$	$1\ 2\ +\ 3\ *$

3 \* +

СТЕК

2

1



# Stack

Вычислить значение выражения в постфиксной форме записи

$1 + 2 * 3$	$(1 + 2) * 3$
$1\ 2\ 3\ *\ +$	$1\ 2\ +\ 3\ *$

\* +

СТЕК

3

2

1



# Stack

Вычислить значение выражения в постфиксной форме записи

$1 + 2 * 3$	$(1 + 2) * 3$
$1\ 2\ 3\ *\ +$	$1\ 2\ +\ 3\ *$

$*$  +

СТЕК

3

2

1



# Stack

Вычислить значение выражения в постфиксной форме записи

$1 + 2 * 3$	$(1 + 2) * 3$
$1\ 2\ 3\ *\ +$	$1\ 2\ +\ 3\ *$

$*$  +

СТЕК

3

2

1



# Stack

Вычислить значение выражения в постфиксной форме записи

$1 + 2 * 3$	$(1 + 2) * 3$
$1\ 2\ 3\ *\ +$	$1\ 2\ +\ 3\ *$

$*$  +

СТЕК

3

2

1





# Stack

Вычислить значение выражения в постфиксной форме записи

$1 + 2 * 3$	$(1 + 2) * 3$
$1\ 2\ 3\ *\ +$	$1\ 2\ +\ 3\ *$

+

СТЕК

6

1



# Stack

Вычислить значение выражения в постфиксной форме записи

$1 + 2 * 3$

$(1 + 2) * 3$

$1\ 2\ 3\ *\ +$

$1\ 2\ +\ 3\ *$

+

СТЕК

6

1



# Stack

Вычислить значение выражения в постфиксной форме записи

$1 + 2 * 3$	$(1 + 2) * 3$
$1\ 2\ 3\ *\ +$	$1\ 2\ +\ 3\ *$

+

СТЕК

6

1



# Stack

Вычислить значение выражения в постфиксной форме записи

$1 + 2 * 3$	$(1 + 2) * 3$
$1\ 2\ 3\ *\ +$	$1\ 2\ +\ 3\ *$

СТЕК

7






# ИТОГИ

Долго, много и не все  
нужно



**Спасибо**   
**за внимание**

