Algorithmen und Datenstrukturen

Prof. Dr. Thomas Wiemann - FB AI

Hochschule Fulda
University of Applied Sciences

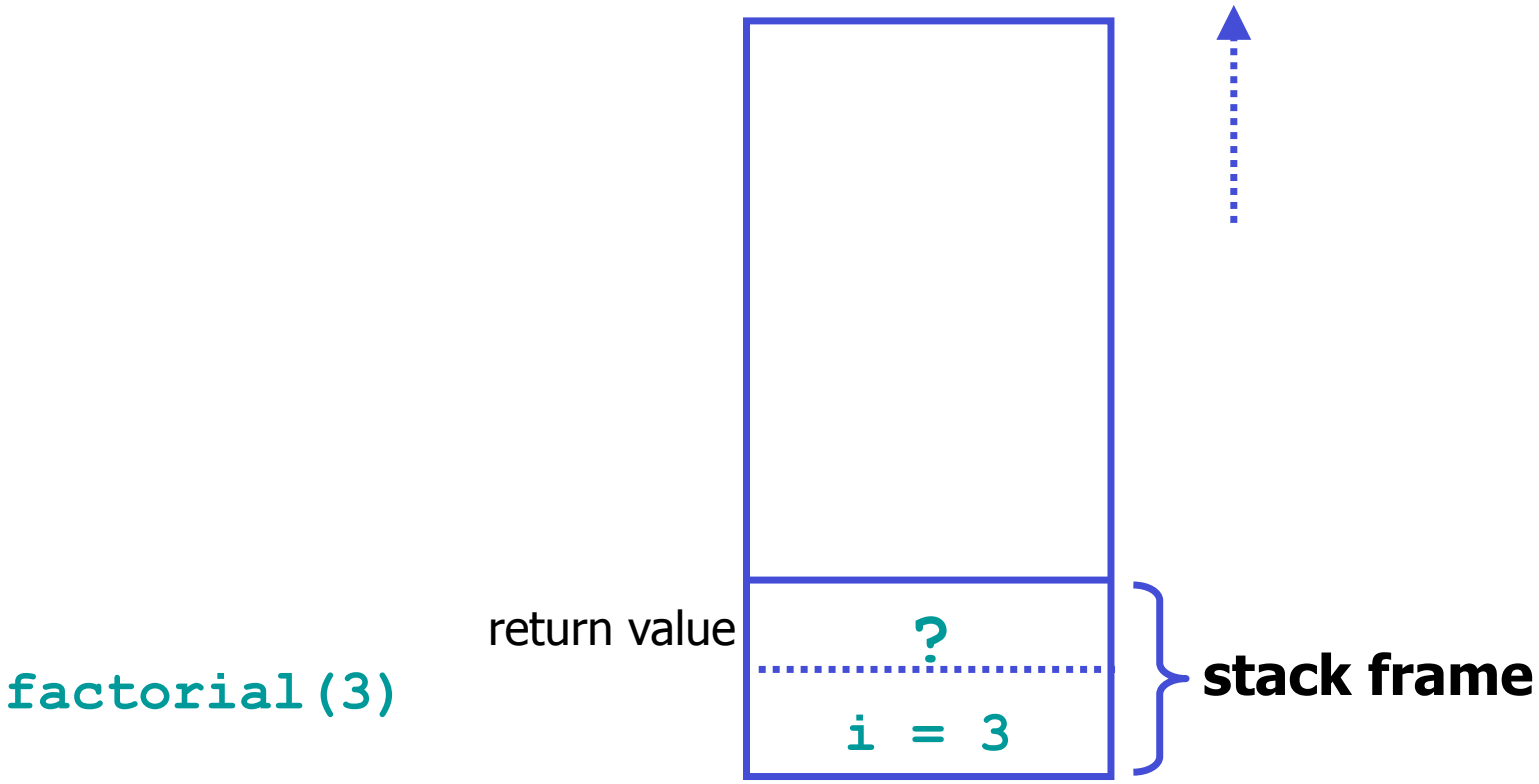
ADT Stack

- Last In - First Out (LIFO) - Prinzip

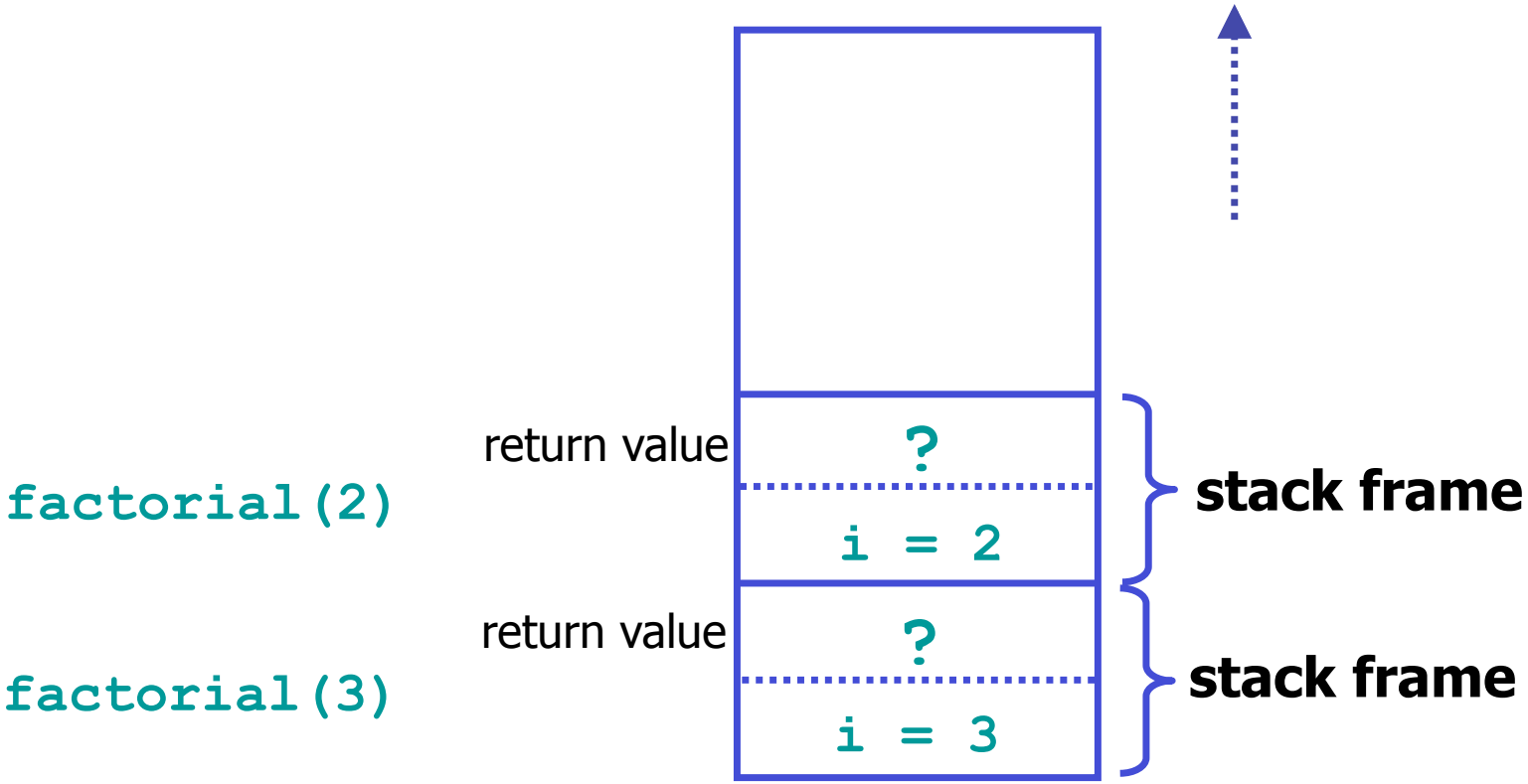
Axiome für ADT Stack

1.) Ein neuer Stack ist leer
2.) Nach `push(x)` ist Stack nicht leer
3.) Nach `push(x)` und `pop()` ist der Stack unverändert
4.) `top()` liefert nach `push(x)` `x`

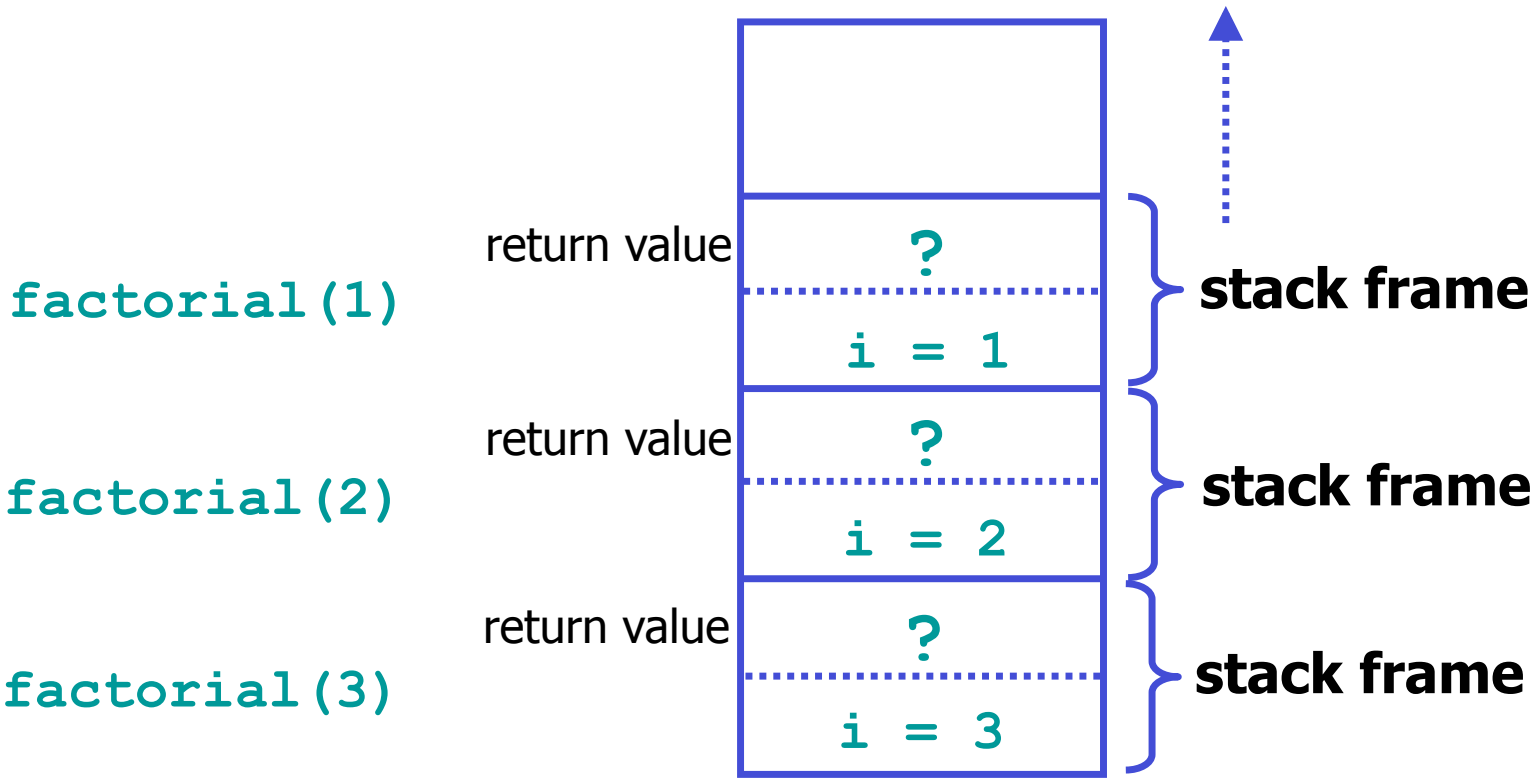
Beispiel Programmstack (1)



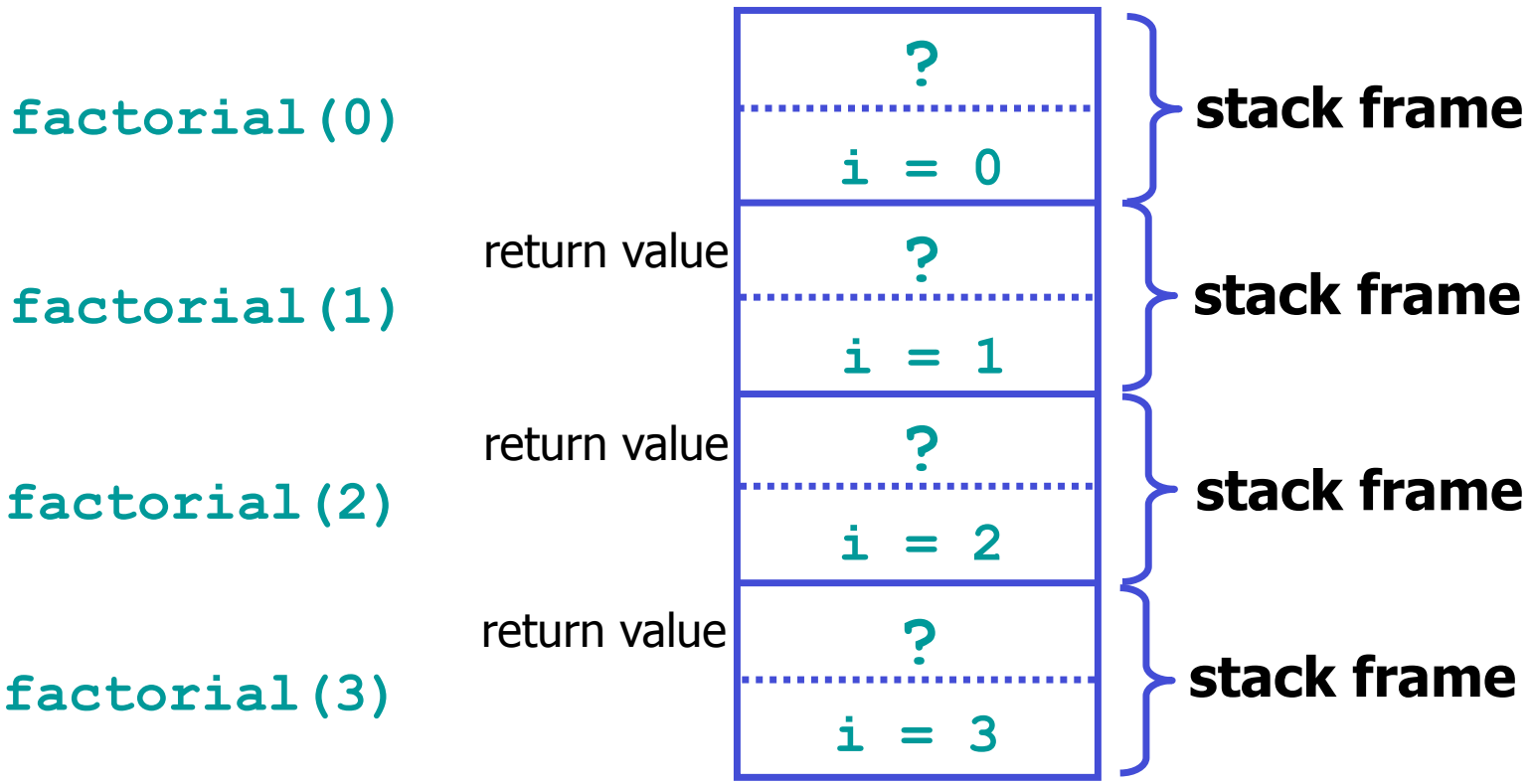
Beispiel Programmstack (2)



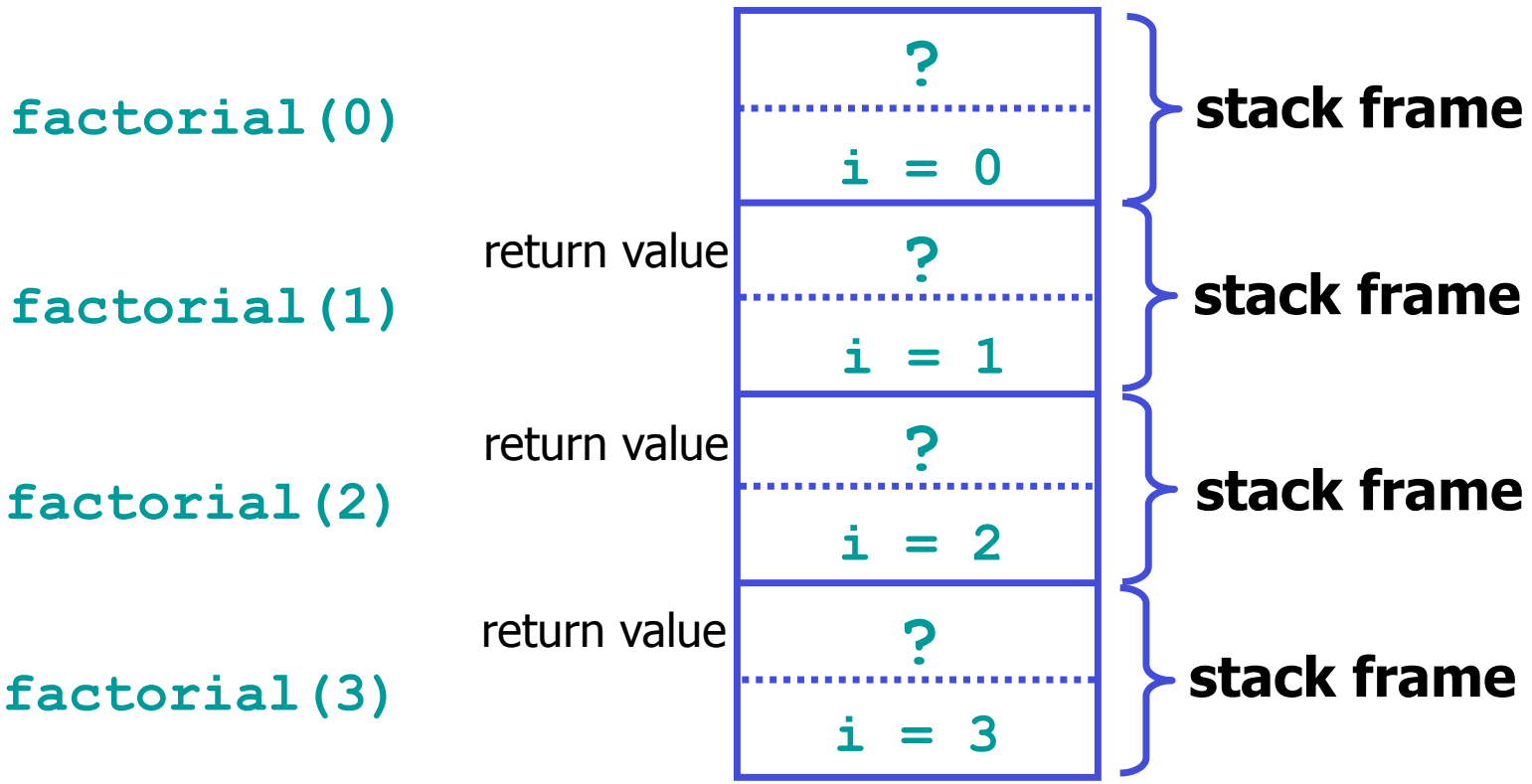
Beispiel Programmstack (3)



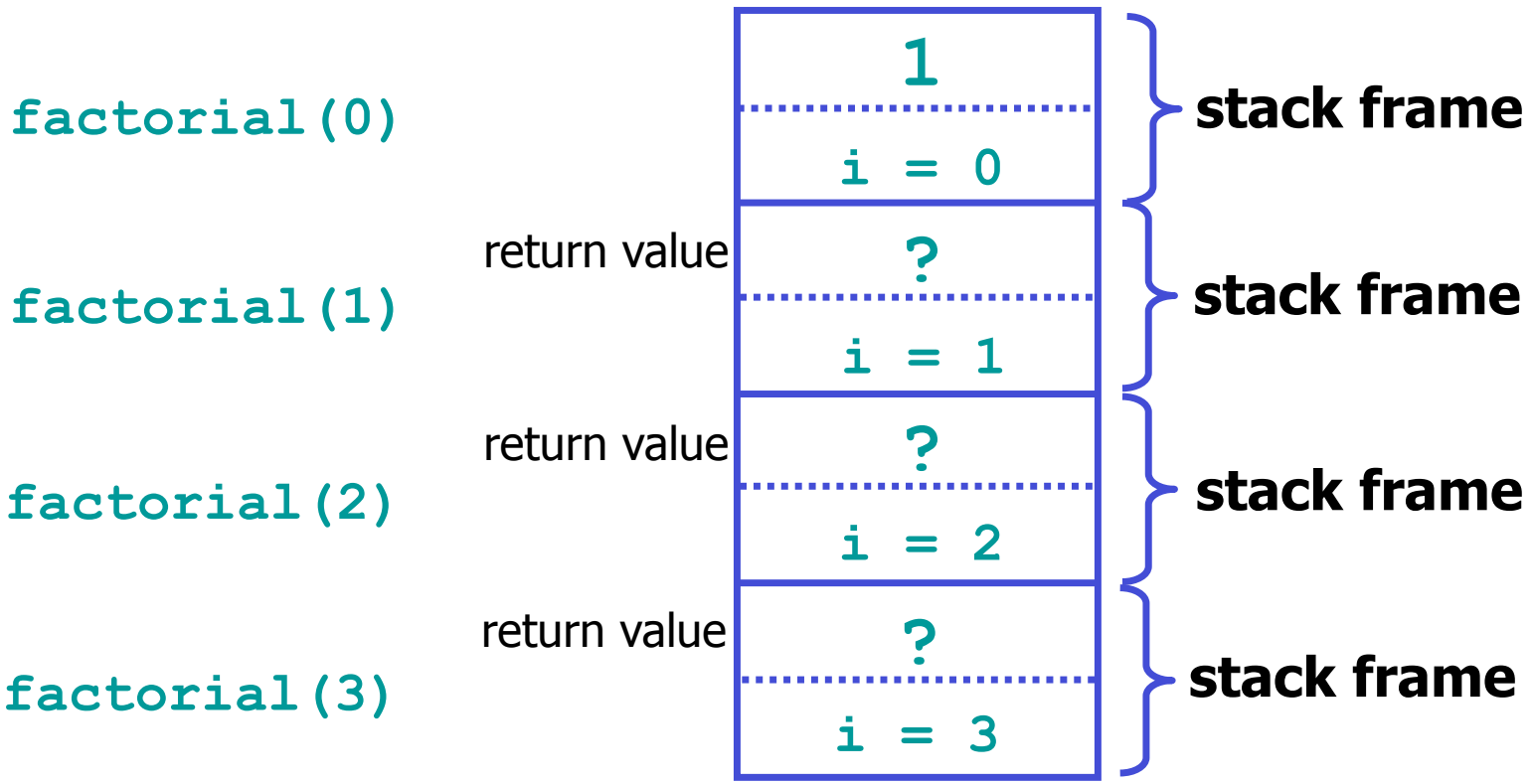
Beispiel Programmstack (4)



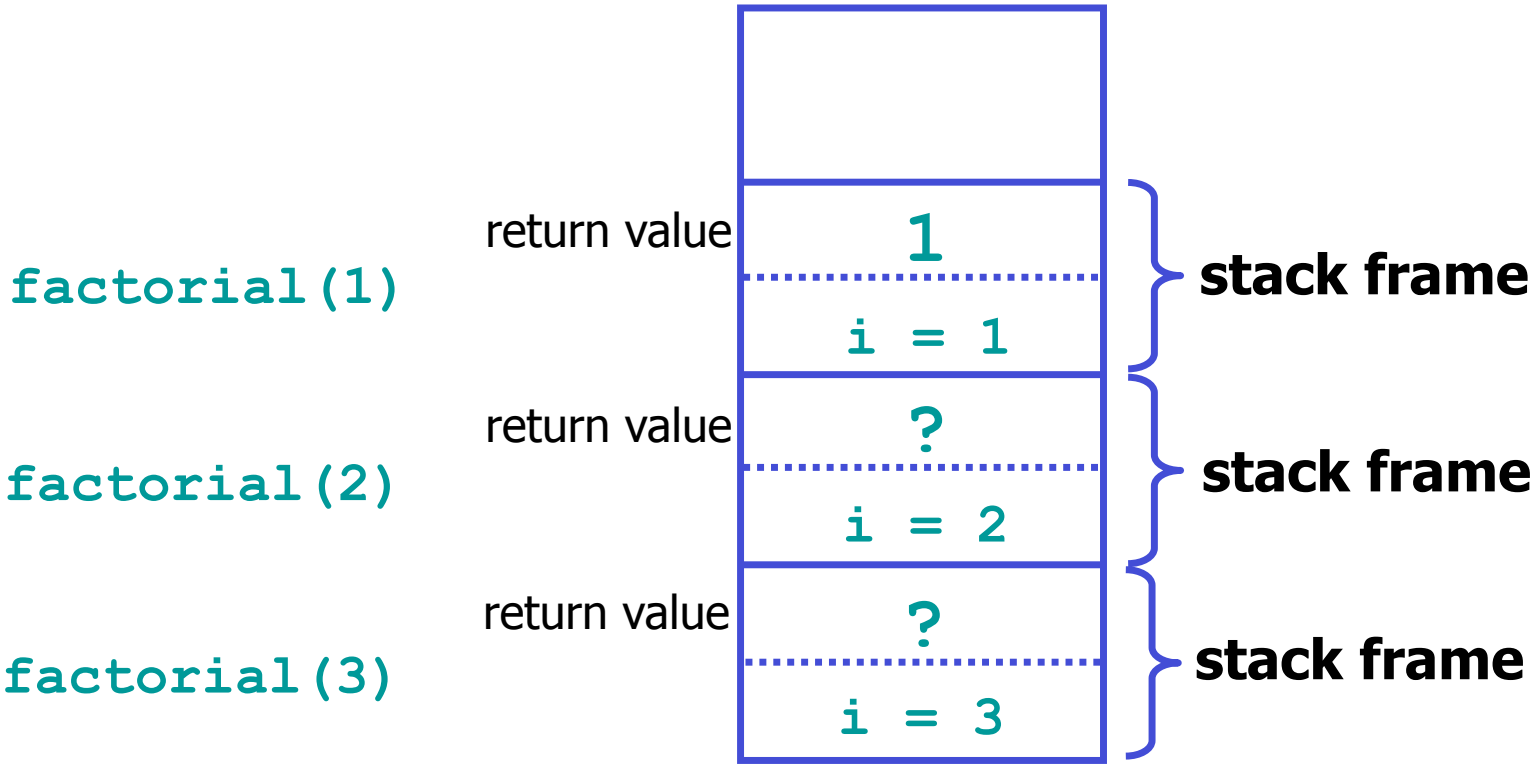
Beispiel Programmstack (5)



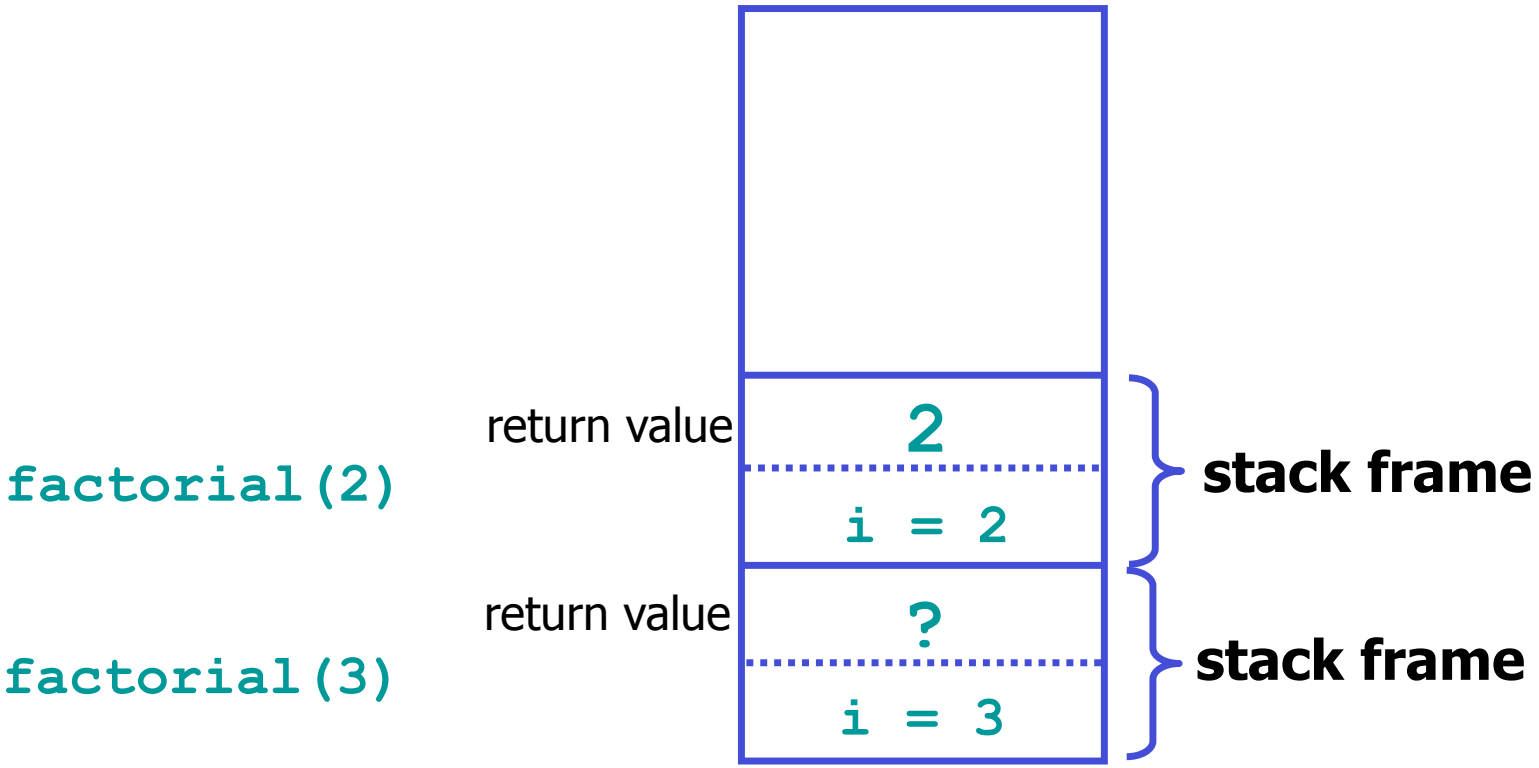
Beispiel Programmstack (6)



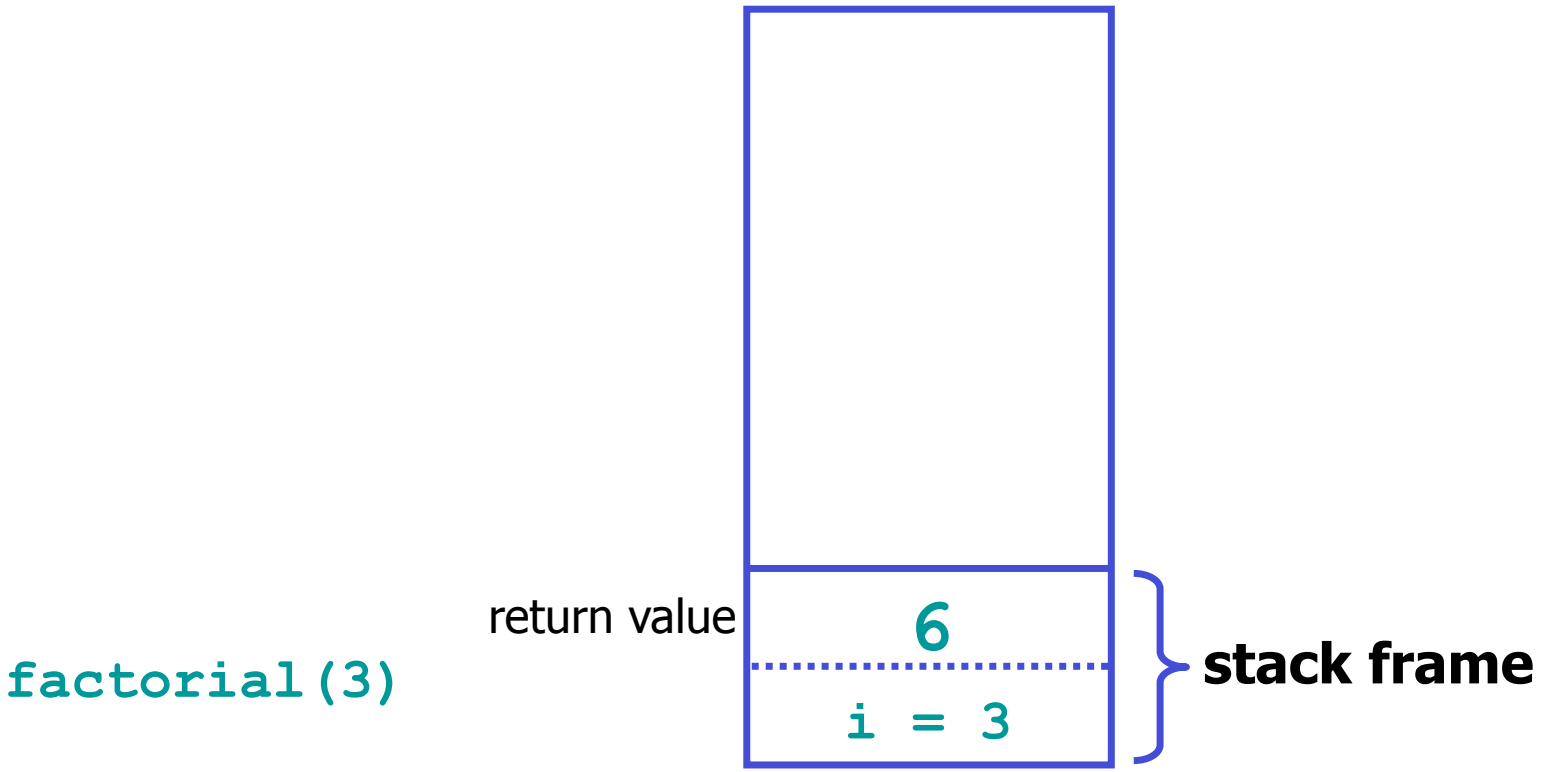
Beispiel Programmstack (7)



Beispiel Programmstack (8)



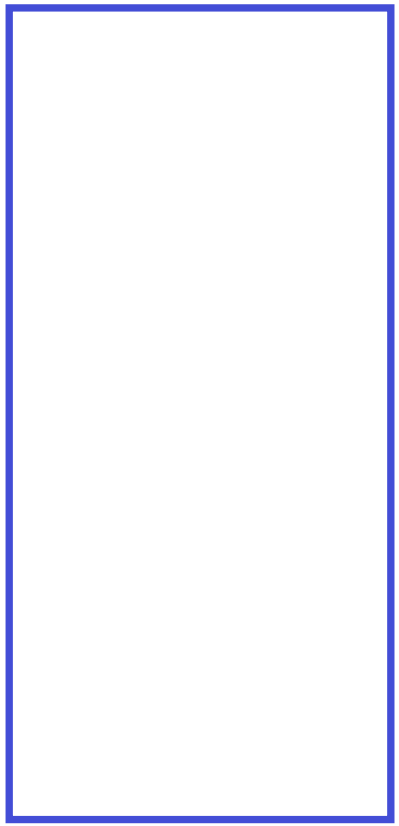
Beispiel Programmstack (9)



Beispiel Programmstack (10)

```
factorial(3)
```

result: 6



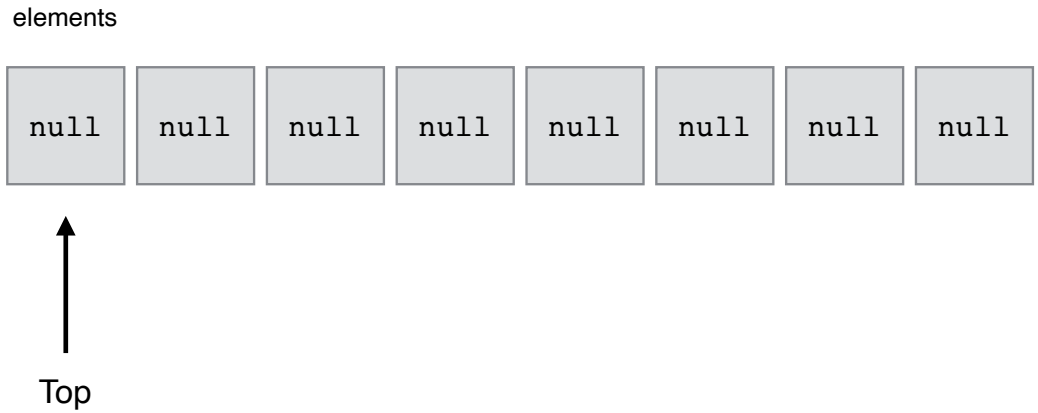
Stack - Vereinfachtes Interface

```
public interface Stack {
    public boolean empty()       // Stack leer oder nicht?
    public void push(Object x)   // Lege x auf den Stapel
    public Object top()          // Liefere oberstes Element
    public void pop()            // Entferne oberstes Element
}
```

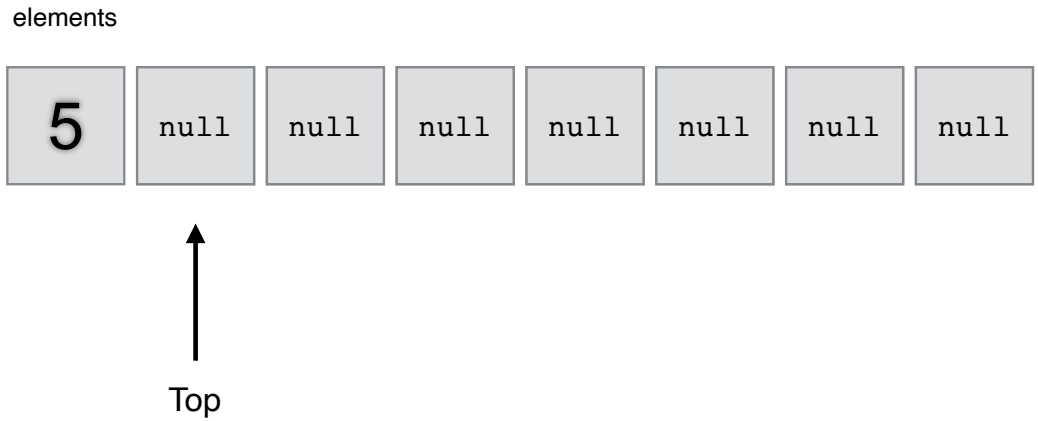
Implementierungen

1.) Mittels Array - dann Stack mit maximaler Höhe
2.) Mittels Verweisen - analog zur Liste

Stack als Array - Push (1)

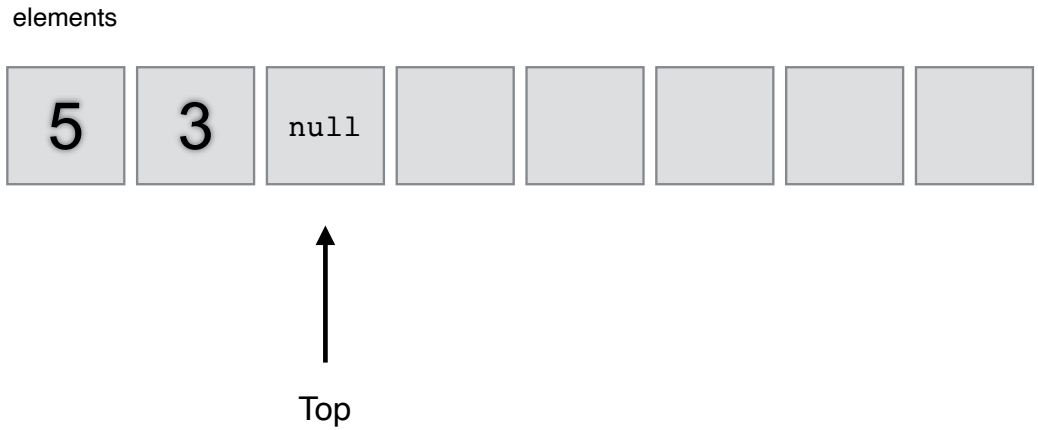


Stack als Array - Push (2)



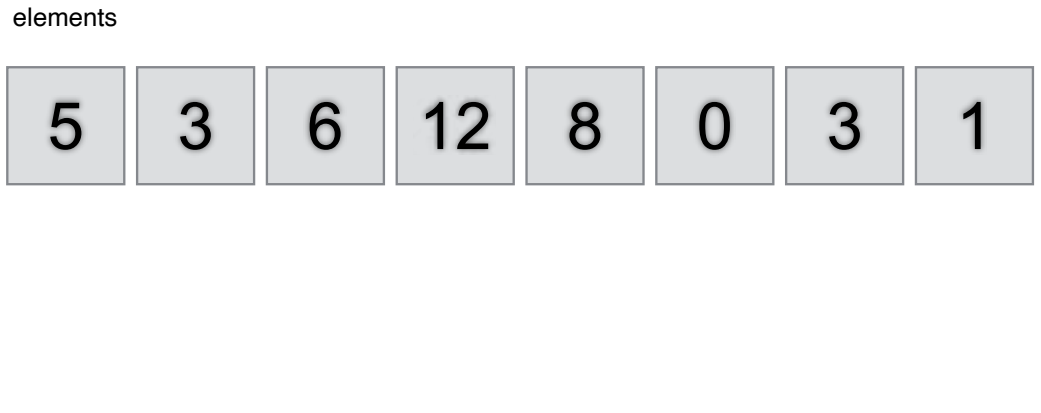

```
s.push(5)
```

Stack als Array - Push (3)




```
s.push(5)
s.push(3)
```

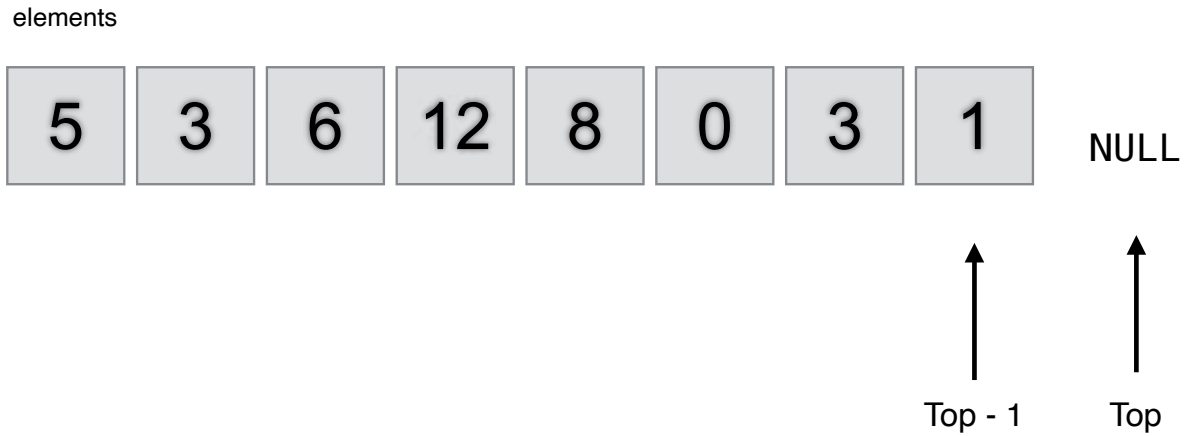
Stack als Array - Push (4)




```
s.push(5)
s.push(3)
s.push(6)
s.push(12)
s.push(8)
s.push(0)
s.push(3)
s.push(1)
```

```
public void push(Object item) {
    if (top == elements.length) {
      throw new IllegalStateException("The stack is full");
    }
    elements[top] = item;
    top++;
}
```

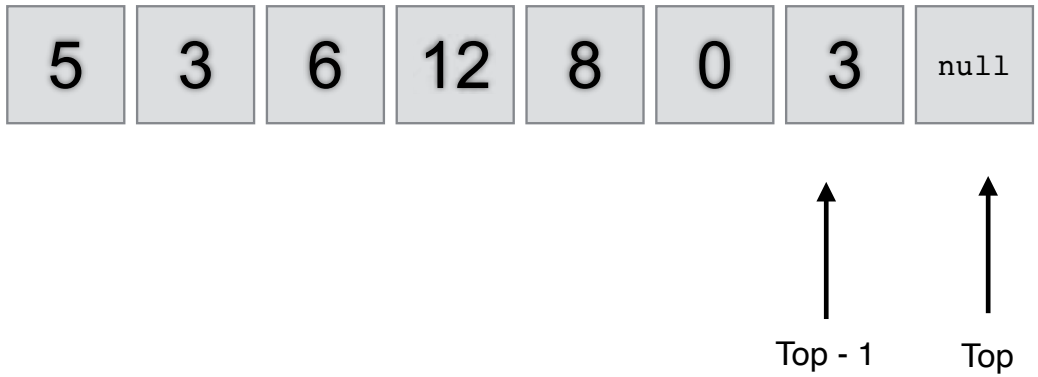
Stack als Array - Top()




```
s.push(5)
s.push(3)
s.push(6)
s.push(12)
s.push(0)
s.push(3)
s.push(1)
s.top()
```

```
public Object top() {
    if (isEmpty()) {
      throw new NoSuchElementException();
    }
    return elements[top - 1];
}
```

Stack als Array - Pop (1)



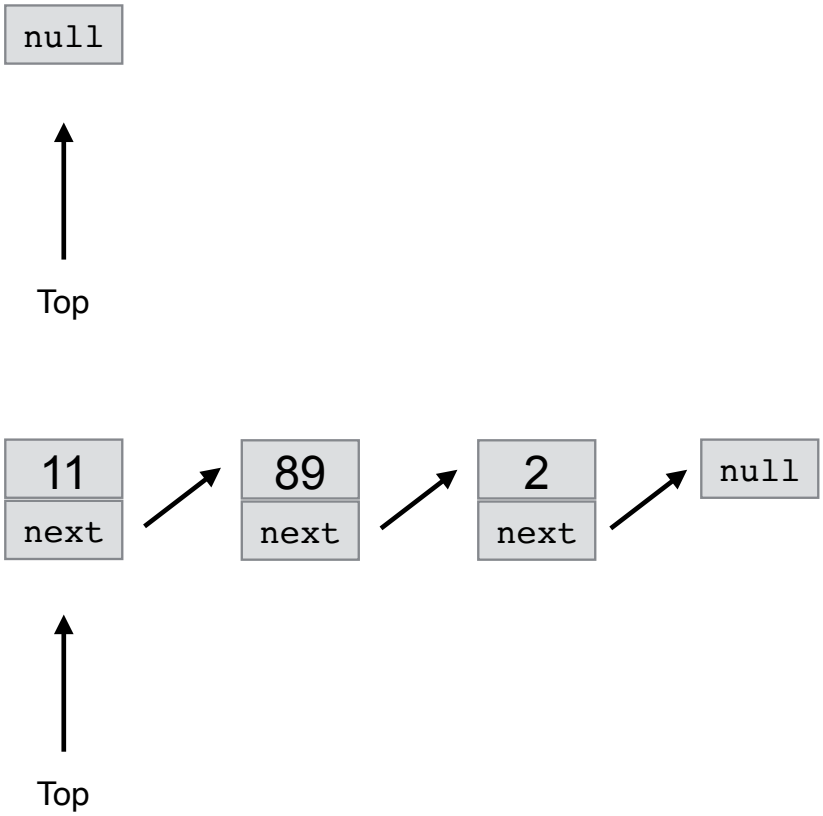

```
s.push(5)
s.push(3)
s.push(6)
s.push(12)
s.push(0)
s.push(3)
s.push(1)
s.top()
s.pop()
```

Stack als Array - Pop (2)


```
s.push(5)
s.push(3)
s.push(6)
s.push(12)
s.push(0)
s.push(3)
s.push(1)
s.peek(1)
s.pop()
s.pop()
s.pop()
s.pop()
s.pop()
s.pop()
s.pop()
```

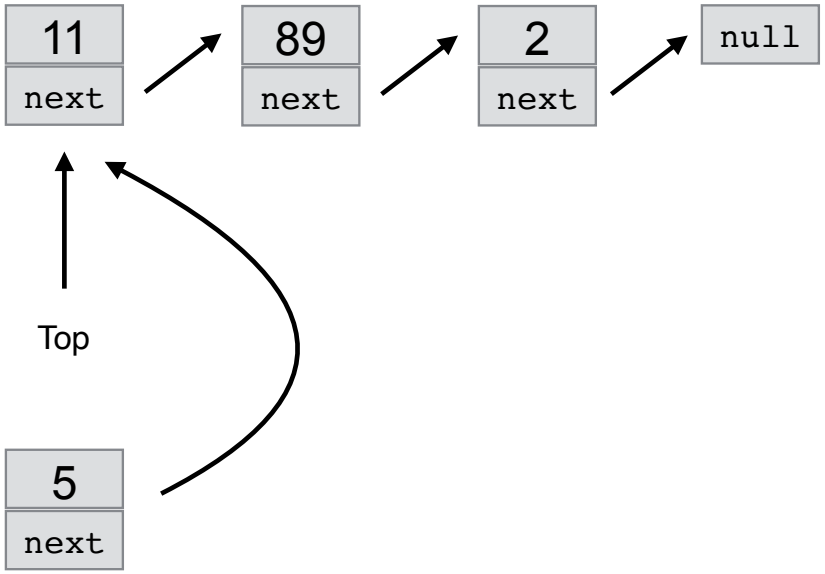
```
public void pop() {
    elements[top - 1] = null;
    top--;
}
```

Stack als einfach verkettete Liste (1)




```
public class LinkedStack implements Stack {
    private Node top;

    public LinkedStack() {
        top = null;
    }

    public boolean empty() {
        return top == null;
    }

    public Object top() {
        if(empty()) throw new
            RuntimeException("Stack Empty");
        return top.data;
    }
```

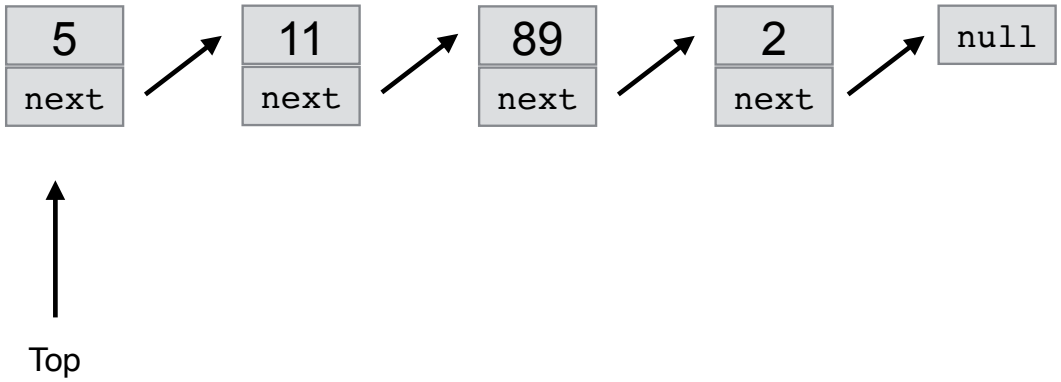
Stack als einfach verkettete Liste - Push



s.push(5)

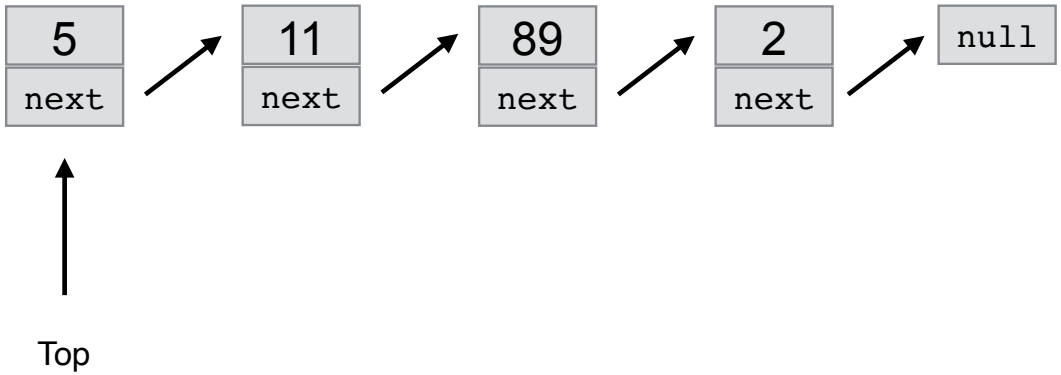
```
public void push(Object x) {
    Node tmp = new Node();
    tmp.data = x;
    tmp.next = top;
    top = tmp;
}
```

Stack als einfach verkettete Liste - Pop




```
s.push(5)
```

Stack als einfach verkettete Liste - Pop




```
s.push(5)
s.pop()
```

```
public void pop() {
  if(empty()) throw new RuntimeException("Stack Empty");
  top = top.next;
}
```

Beispiel: Reverse

```
Stack s = new LinkedStack();
int[] a = (IO.readInts(" "));
for(int i = 0; i < a.length; i++)
    s.push(new Integer(a[i]));

while(!s.empty()) {
    IO.println(((Integer)s.top()).intValue());
    s.pop();
}
```

ADT Queue

ADT Queue

Gegebenenfalls leere Folge von Elementen zusammen mit einen ggf. undefinierten Frontelement. Die Elemente sind nach dem First In - First Out (FIFO) Prinzip angeordnet.

```
public Interface Queue{
    public boolean empty();     // Schlange leer?
    public void enq(Object x); // Neues Objekt hinten anstellen
    public Object front();      // Gib vorderstes Element
    public void deq();          // Entferne vorderstes Element
}
```

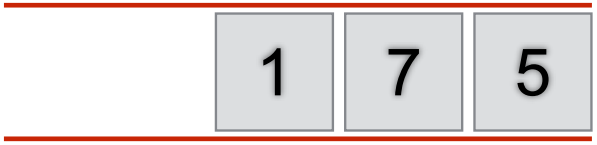
`q.enq(5)`

ADT Queue

```
q.enq(5)
q.enq(7)
```

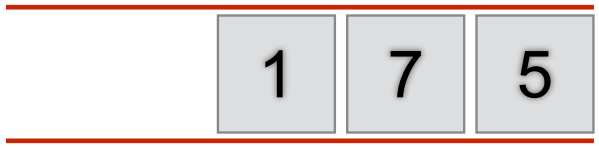
ADT Queue

```
q.enq(5)
q.enq(7)
q.enq(1)
```



ADT Queue

```
q.enq(5)
q.enq(7)
q.enq(1)
q.deq()
```



ADT Queue

```
q.enq(5)
q.enq(7)
q.enq(1)
q.deq()
```

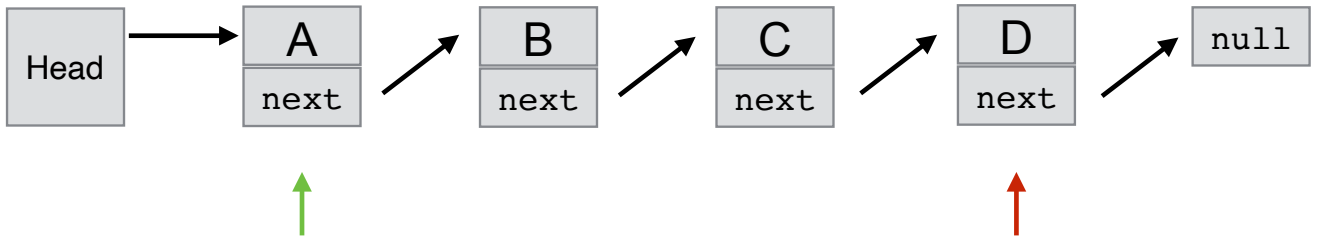
ADT Queue

Schlange ist voll

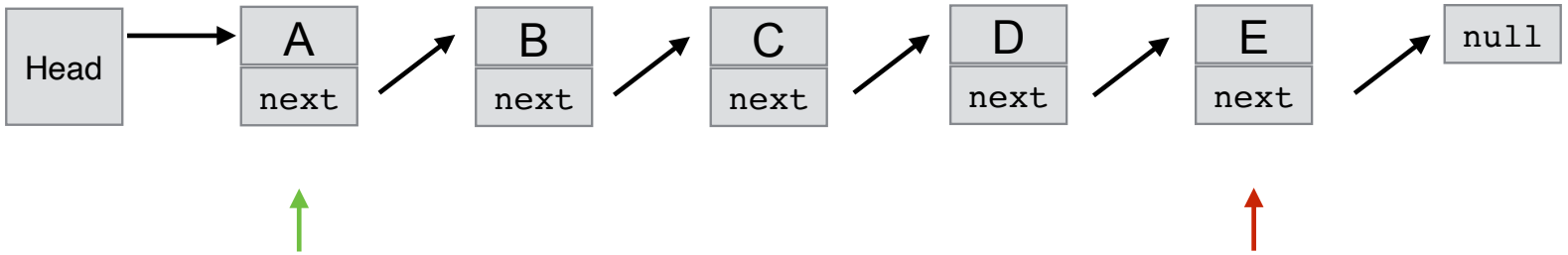
```
q.enq(5)
q.enq(7)
q.enq(1)
q.deq()
q.enq(4)
q.enq(2)
q.enq(9)
```

Queue - Implementierung als Liste

- Implementierung über verkettete Liste

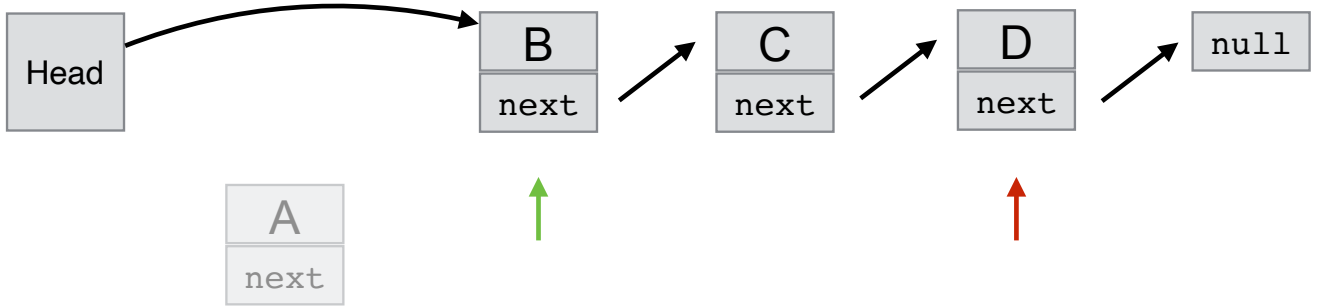


- Enqueue



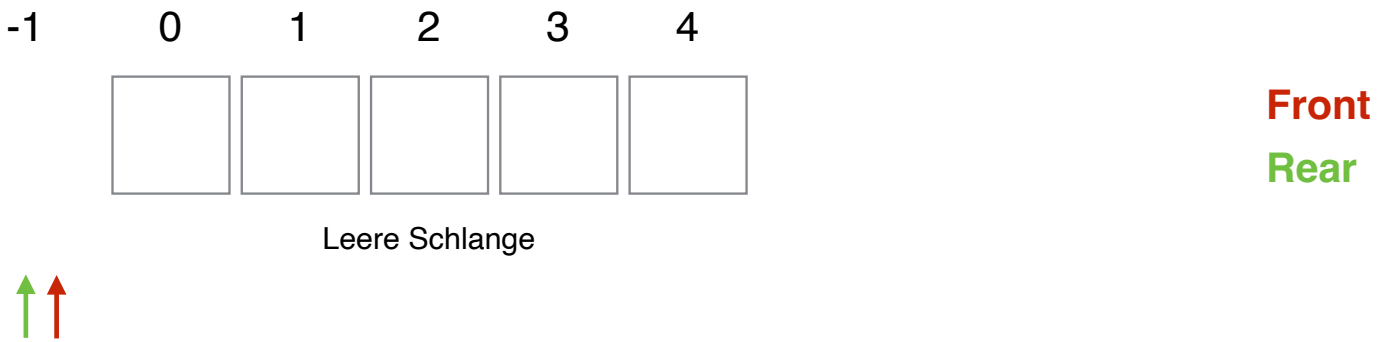
Queue - Implementierung als Liste

- Dequeue:



Queue - Implementierung als Array

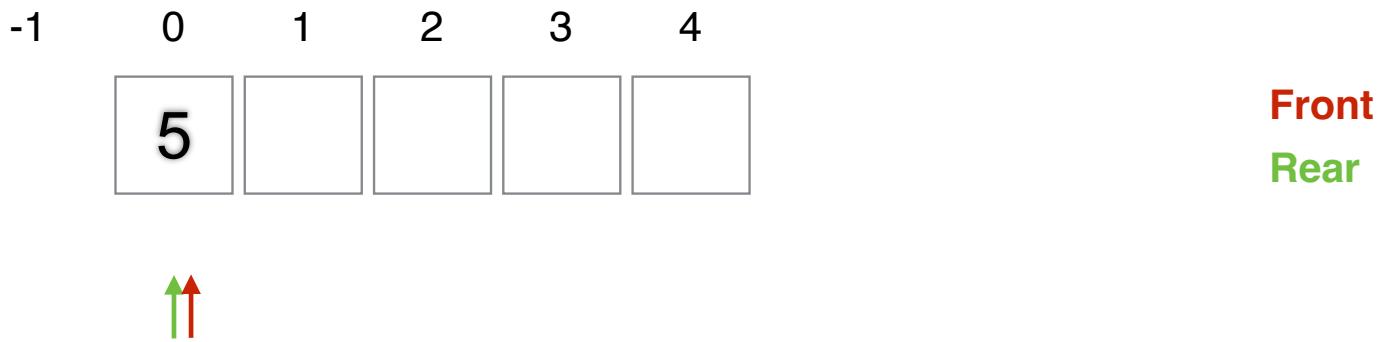
- Verwalte ein Array vorgegebener Größe (maximale Länge der Schlange)
- Zeiger auf vorderes und hinteres Ende der Schlange



Queue - Implementierung als Array

- Verwalte ein Array vorgegebener Größe (maximale Länge der Schlange)
- Zeiger auf vorderes und hinteres Ende der Schlange

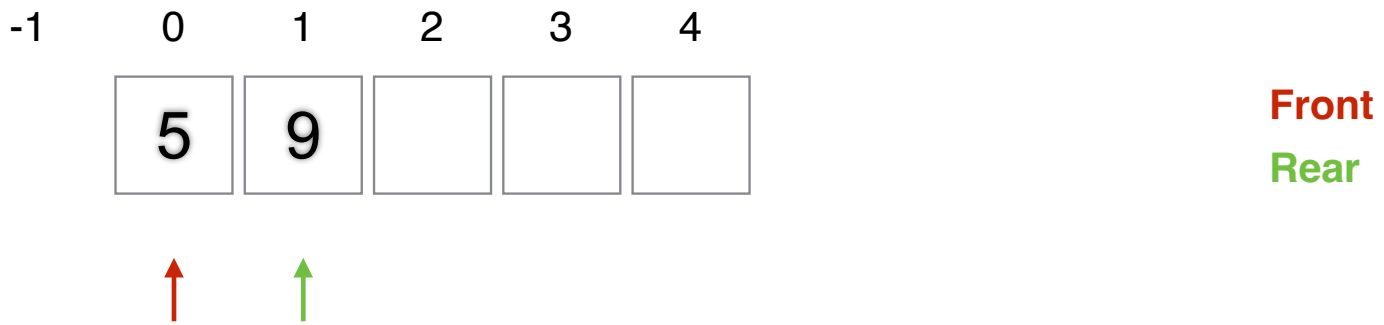
```
q.enq(5)
```



Queue - Implementierung als Array

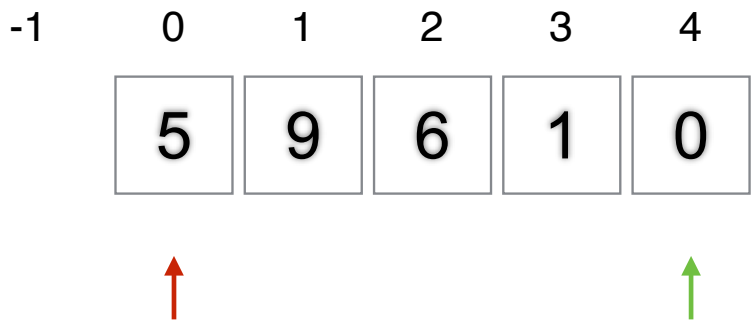
- Verwalte ein Array vorgegebener Größe (maximale Länge der Schlange)
- Zeiger auf vorderes und hinteres Ende der Schlange

```
q.enq(5)
q.enq(9)
```



Queue - Implementierung als Array

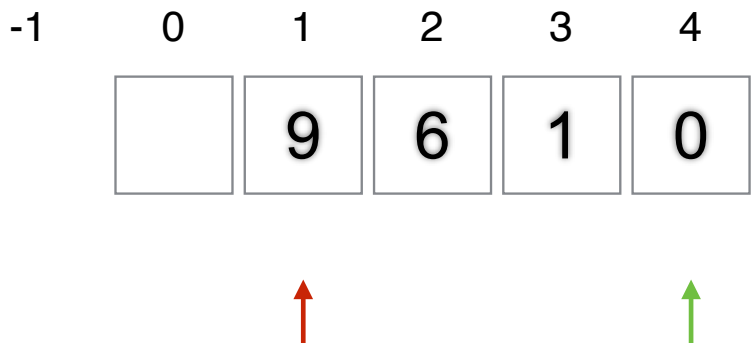
- Verwalte ein Array vorgegebener Größe (maximale Länge der Schlange)
- Zeiger auf vorderes und hinteres Ende der Schlange




```
q.enq(5)
q.enq(9)
q.enq(6)
q.enq(1)
q.enq(0)
```

Queue - Implementierung als Array

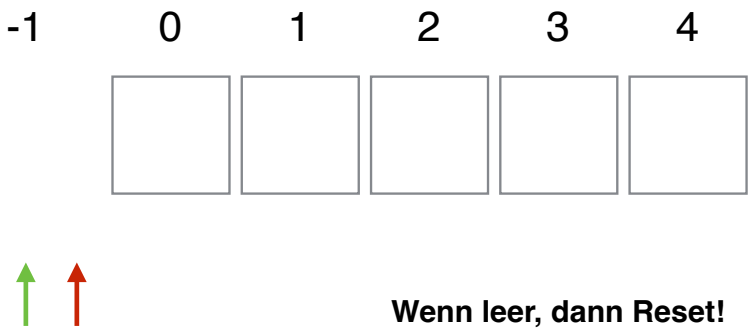
- Verwalte ein Array vorgegebener Größe (maximale Länge der Schlange)
- Zeiger auf vorderes und hinteres Ende der Schlange




```
q.enq(5)
q.enq(9)
q.enq(6)
q.enq(1)
q.enq(0)
q.deq()
```

Queue - Implementierung als Array

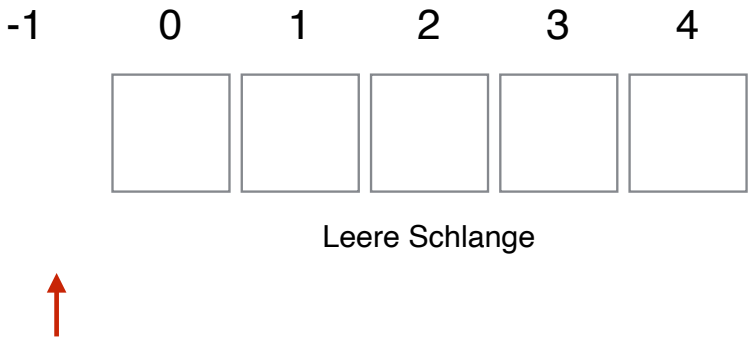
- Verwalte ein Array vorgegebener Größe (maximale Länge der Schlange)
- Zeiger auf vorderes und hinteres Ende der Schlange



Wenn leer, dann Reset!

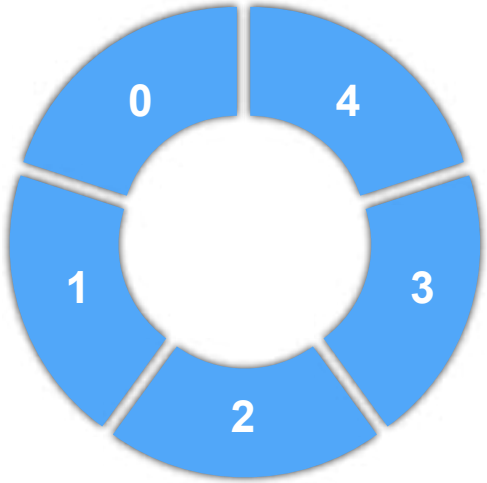
```
q.enq(5)
q.enq(9)
q.enq(6)
q.enq(1)
q.enq(0)
q.deq()
q.deq()
q.deq()
q.deq()
q.deq()
```

Circular Queue



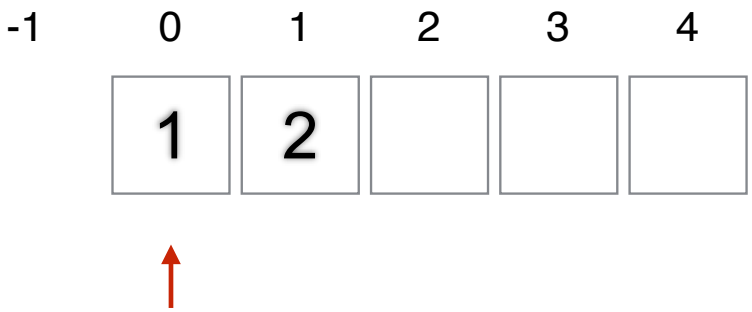
Head

Count = 0



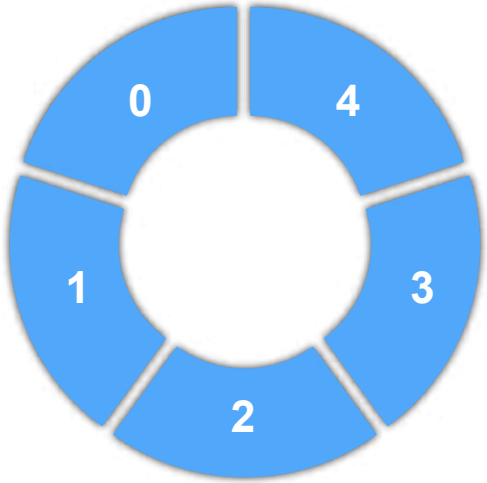
Circular Queue

```
q.enq(1)

q.enq(2)
```



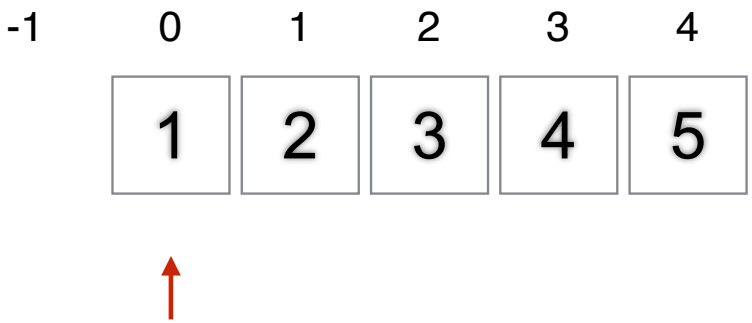
Head

Count = 2



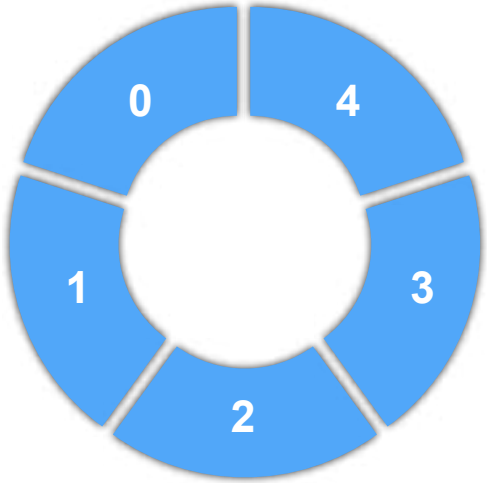
Circular Queue

```
q.enq(1)
q.enq(2)
q.enq(3)
q.enq(4)
q.enq(5)
```



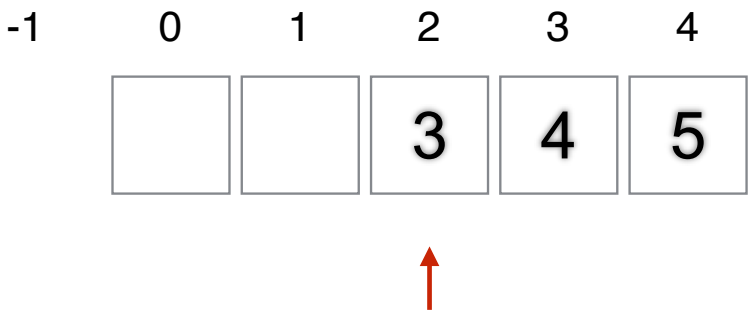
Head

Count = 5



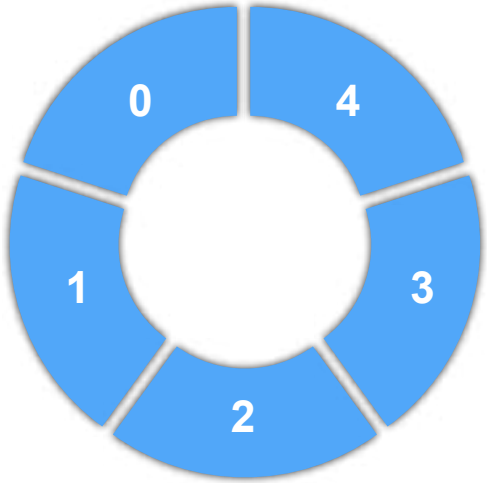
Circular Queue

```
q.enq(1)
q.enq(2)
q.enq(3)
q.enq(4)
q.enq(5)
q.deq()
q.deq()
```



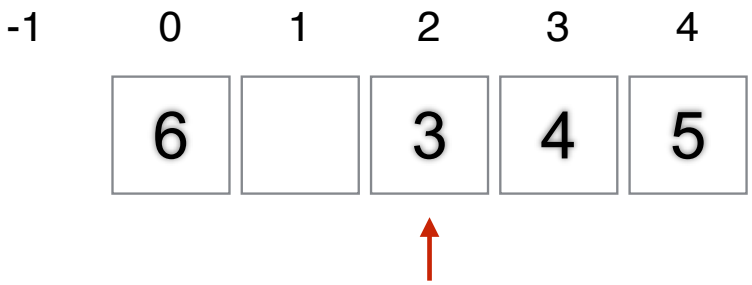
Head

Count = 3



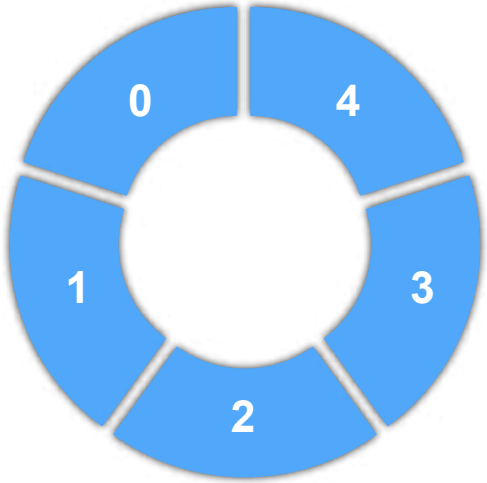
Circular Queue

```
q.enq(1)
q.enq(2)
q.enq(3)
q.enq(4)
q.enq(5)
q.deq()
q.deq()
q.enq(6)
```



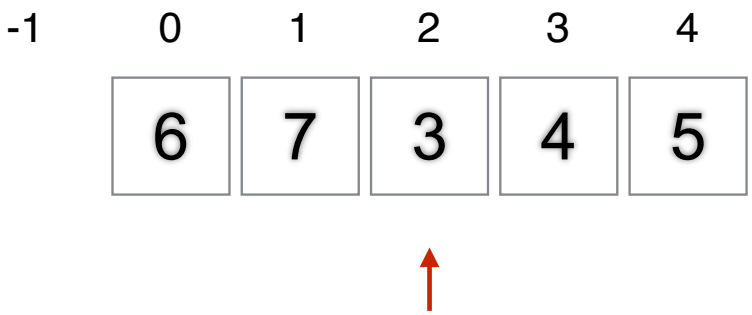
Front

Count = 4



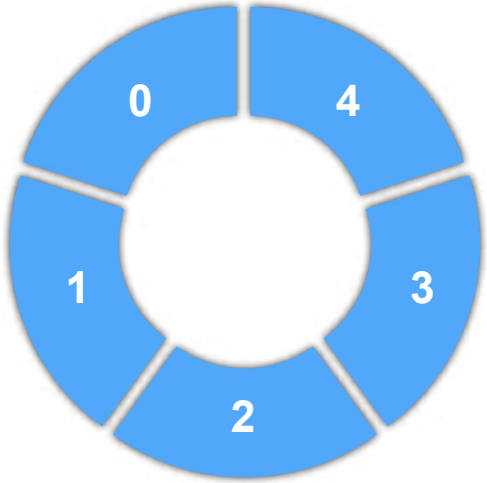
Circular Queue

```
q.enq(1)
q.enq(2)
q.enq(3)
q.enq(4)
q.enq(5)
q.deq()
q.deq()
q.enq(6)
q.enq(7)
```



Head

Count = 5



Schlange voll

Queue als Array - Java

```
class ArrayQueue implements Queue {
    private Object[] buffer;
    private int head, count;

    public ArrayQueue(int N) {
        buffer = new Object[N];
        count = rear = 0;
    }

    private boolean full() {
        return count == buffer.length;
    }

    public boolen empty() {
        return count == 0;
    }
```

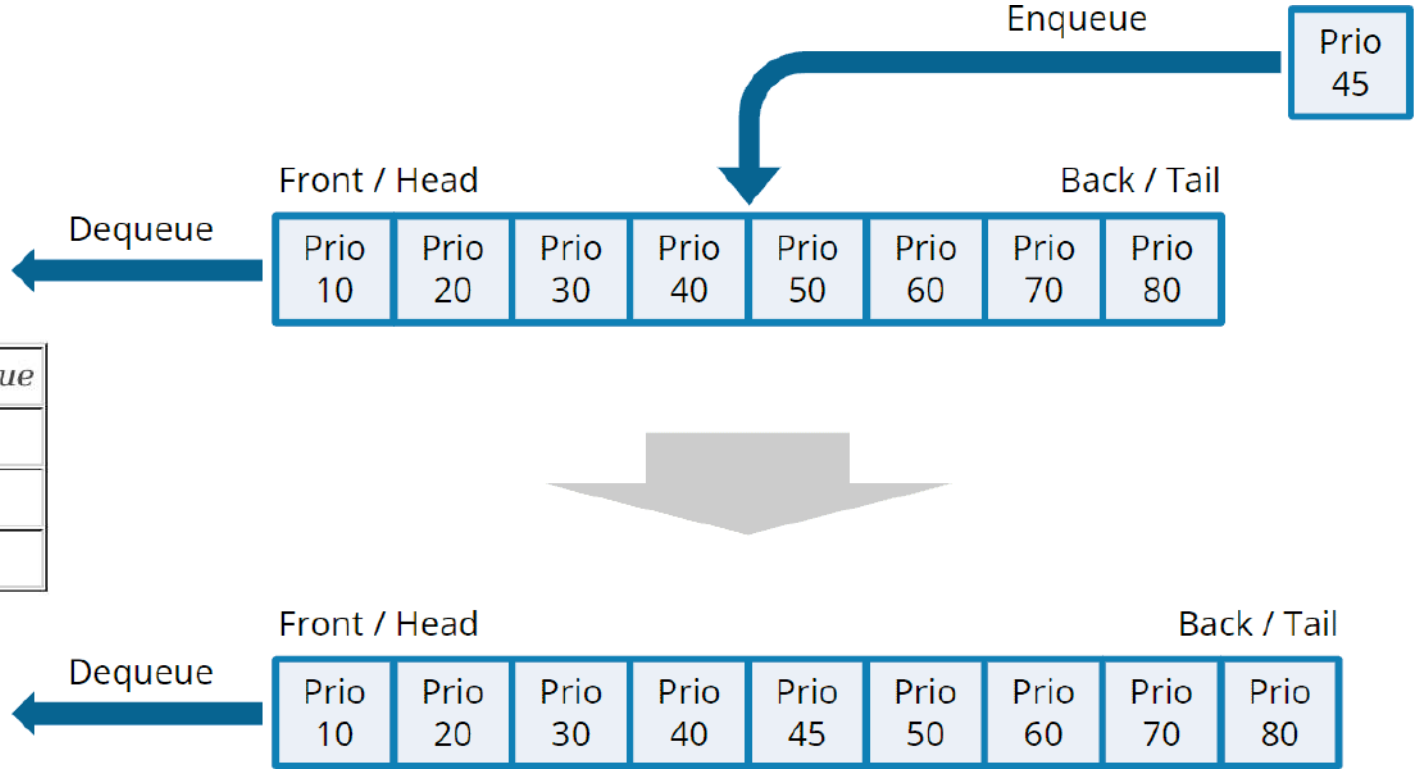
```
    public void enq(Object x) {
        if(full()) throw new ...
        buffer[(head + count) % buffer.length] = x;
        count++;
    }

    public Object front() {
        if(empty()) throw new ...
        return buffer[head];
    }

    public void deq() {
        if(empty()) throw new ...
        buffer[head] = null;
        head = (head + 1) % buffer.length;
        count--;
    }
```

Beispiel in “echtem” Java

```java
import java.util.Scanner;
import java.util.Queue;
import java.util.LinkedList;

public class Mystery {
    private Queue<String> q;

    public void do_something(int n) {
        q = new LinkedList<String>();
        q.offer("1");
        for (int i = 0; i < n; i++) {
            String front = q.remove();
            System.out.println((i+1)+": "+front);
            q.offer(front+"0");
            q.offer(front+"1");
        }
    }

    public static void main(String[] args) {
        System.out.println("Do something to what decimal value?");
        Scanner scanner = new Scanner(System.in);
        int n = scanner.nextInt();
        scanner.close();
        Mystery m = new Mystery();
        m.do_something(n);
    }
}
```

Priority Queue

- Alle Elemente haben einen Schlüssel
- Sorge dafür, das immer das Element mit dem größten / kleinsten Schlüssel vorne steht
- “Drängelschlange”



	Throws exception	*Returns special value*
Insert	`add(e)`	`offer(e)`
Remove	`remove()`	`poll()`
Examine	`element()`	`peek()`

https://www.happycoders.eu/de/algorithmen/priorityqueue-java/

Priority Queue

```java
public class PriorityQueueExample {
  public static void main(String[] args) {
    Queue<Integer> queue = new PriorityQueue<>();

    // Enqueue random numbers
    for (int i = 0; i < 8; i++) {
      int element = ThreadLocalRandom.current().nextInt(100);
      queue.offer(element);
      System.out.printf("queue.offer(%2d)    -->  queue = %s%n", element, queue);
    }

    // Dequeue all elements
    while (!queue.isEmpty()) {
      Integer element = queue.poll();
      System.out.printf("queue.poll() = %2d  -->  queue = %s%n", element, queue);
    }
  }
}
```

https://www.happycoders.eu/de/algorithmen/priorityqueue-java/

Priority Queue

- Kleinstes Element immer vorne!
- Nicht zwangsläufig komplett sortiert
- Heap-Repräsentation des Arrays ➠Siehe Abschnitt 5

```
queue.offer(80)    -->  queue = [80]
queue.offer(14)    -->  queue = [14, 80]
queue.offer(10)    -->  queue = [10, 80, 14]
queue.offer(50)    -->  queue = [10, 50, 14, 80]
queue.offer( 9)    -->  queue = [9, 10, 14, 80, 50]
queue.offer(58)    -->  queue = [9, 10, 14, 80, 50, 58]
queue.offer(41)    -->  queue = [9, 10, 14, 80, 50, 58, 41]
queue.offer( 1)    -->  queue = [1, 9, 14, 10, 50, 58, 41, 80]
queue.poll() =  1  -->  queue = [9, 10, 14, 80, 50, 58, 41]
queue.poll() =  9  -->  queue = [10, 41, 14, 80, 50, 58]
queue.poll() = 10  -->  queue = [14, 41, 58, 80, 50]
queue.poll() = 14  -->  queue = [41, 50, 58, 80]
queue.poll() = 41  -->  queue = [50, 80, 58]
queue.poll() = 50  -->  queue = [58, 80]
queue.poll() = 58  -->  queue = [80]
queue.poll() = 80  -->  queue = []
```

Dequeue

- Double-Ended Queue
- Erlaubt einfügen am Anfang und am Ende
- Kein FIFO-Prinzip mehr
- `addRear` und `addFront` sowie `removeRear` und `removeFront`

Comparable Interface

```
public interface Comparable {
    public int compareTo(Object x)
    {
        // returns 0 if this == x
        // returns < 0 if this < 0
        // returns > 0 if this > 0
    }
}
```

```
public class StudentComparable implements Comparable {
    private int matNr;
    public int compareTo(Object x)
    {
        if(!x.instanceOf(StudentComparable)) throw new RuntimeException();
        return this.matNr - ((StudentComparable)x).matNr;
    }
}
```

Set

- Repräsentation von (math.) Mengen
- Ordnung auf den Objekten erforderlich (Comparable)
- Pseudo-Java Interface:

```
public interface Set {
    public boolean empty();
    public Comparable find(Comparable x);
    public boolean insert(Comparable x);
    public boolean delete(Comparable x);
}
```

Set - Vereinigung in Java

```
public static void main(String args[]) {
   HashSet <String> set1 = new HashSet <String>();
   HashSet <String> set2 = new HashSet <String>();
   set1.add("Mat");
   set1.add("Sat");
   set1.add("Cat");

   System.out.println("Set1 = "+ set1);
   set2.add("Mat");
   set2.add("Cat");
   set2.add("Fat");
   set2.add("Hat");

   System.out.println("Set2 = "+ set2);
   set1.addAll(set2);
   System.out.println("Union = "+ set1);
}
```

```
Set1 = [Mat, Sat, Cat]
Set2 = [Mat, Cat, Fat, Hat]
Union = [Mat, Sat, Cat, Fat, Hat]
```

Mengenoperationen - Schnittmenge

```
public static void main(String args[]) {
      HashSet <String> set1 = new HashSet <String>();
      HashSet <String> set2 = new HashSet <String>();
      set1.add("Mat");
      set1.add("Sat");
      set1.add("Cat");
      System.out.println("Set1 = "+ set1);
      set2.add("Mat");
      set2.add("Cat");
      set2.add("Fat");
      set2.add("Hat");
      System.out.println("Set2 = "+ set2);
      set1.retainAll(set2);
      System.out.println("Intersection = "+ set1);
 }
```

Map

- Mapping vom Key / Value Paaren
- Pseudo-Java Interface

```
int size()
boolean isEmpty()
boolean containsKey(Object key)
boolean containsValue(Object value)
Object get(Object key)
Object put(Object key, Object value)
Object remove(Object key)
boolean equals(Object o)
```

Map - Beispiel

```
public static void main(String[] args) {
        Map<String, Integer> vehicles = new HashMap<>();

        // Add some vehicles.
        vehicles.put("BMW", 5);
        vehicles.put("Mercedes", 3);
        vehicles.put("Audi", 4);
        vehicles.put("Ford", 10);

        System.out.println("Total vehicles: " + vehicles.size());

        // Iterate over all vehicles, using the keySet method.
        for (String key : vehicles.keySet())
            System.out.println(key + " - " + vehicles.get(key));
        System.out.println();

        String searchKey = "Audi";
        if (vehicles.containsKey(searchKey))
            System.out.println("Found total " + vehicles.get(searchKey) + " " + searchKey + " cars!\n");

        // Clear all values.
        vehicles.clear();

        // Equals to zero.
        System.out.println("After clear operation, size: " + vehicles.size());
    }
```

Konkrete Implementierung: HashMap. Siehe nächstes Kapitel!

ADTs in Java

- Ausschnitt aus der Java Collections API Documentation
- Collections:
  - java.util.Set
  - java.util.SortedSet
  - java.util.NavigableSet
  - java.util.Queue
  - java.util.concurrent.BlockingQueue
  - java.util.concurrent.TransferQueue
  - java.util.Deque
  - java.util.concurrent.BlockingDeque
- Maps:
  - java.util.SortedMap
  - java.util.NavigableMap
  - java.util.concurrent.ConcurrentMap
  - java.util.concurrent.ConcurrentNavigableMap

		Implementations				
		Hash Table	**Resizable Array**	**Balanced Tree**	**Linked List**	**Hash Table + Linked List**
Interfaces	**Set**	HashSet		TreeSet		LinkedHashSet
	List		ArrayList		LinkedList	
	Deque		ArrayDeque		LinkedList	
	Map	HashMap		TreeMap		LinkedHashMap