Algoritmos e Estruturas de Dados

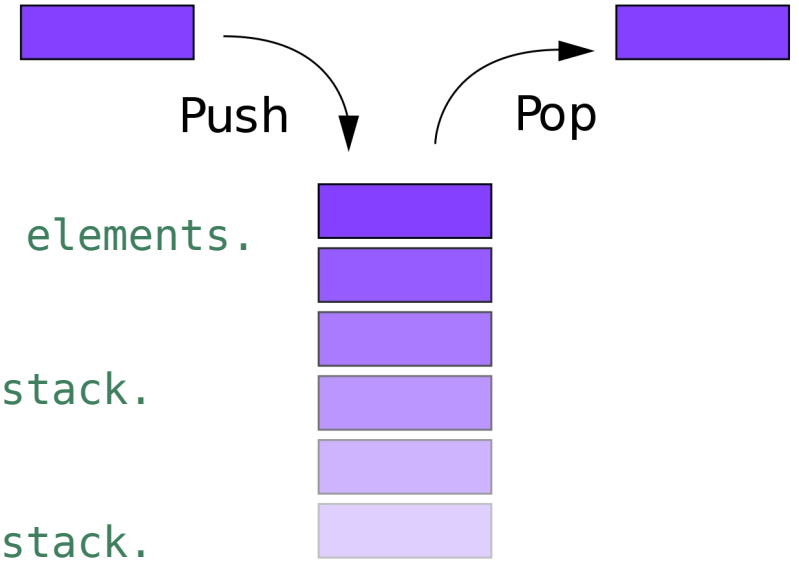
TAD **Stack** – Capítulo 6

2019/20

Package dataStructures: interfaces (TADs)

- Pilha (interface ***Stack***)
  - Coleção de elementos com disciplina LIFO (last-in-first-out)

TAD ***Stack<E>***

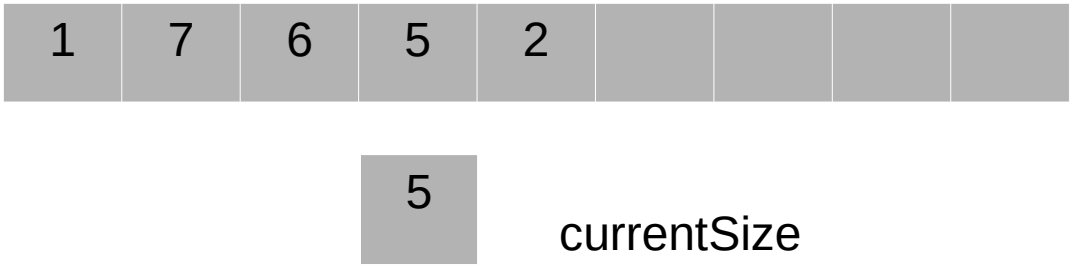



```
package dataStructures;

public interface Stack<E> {
    // Returns true iff the stack contains no elements.
    boolean isEmpty( );

    // Returns the number of elements in the stack.
    int size( );

    // Returns the element at the top of the stack.
    //@throws NoElementException if isEmpty()
    E top( ) throws NoElementException;

    // Inserts the specified element onto the top of the stack.
    void push( E element );

    // Removes and returns the element at the top of the stack.
    //@throws NoElementException if isEmpty()
    E pop( ) throws NoElementException;
}
```

TAD **Stack<E>**

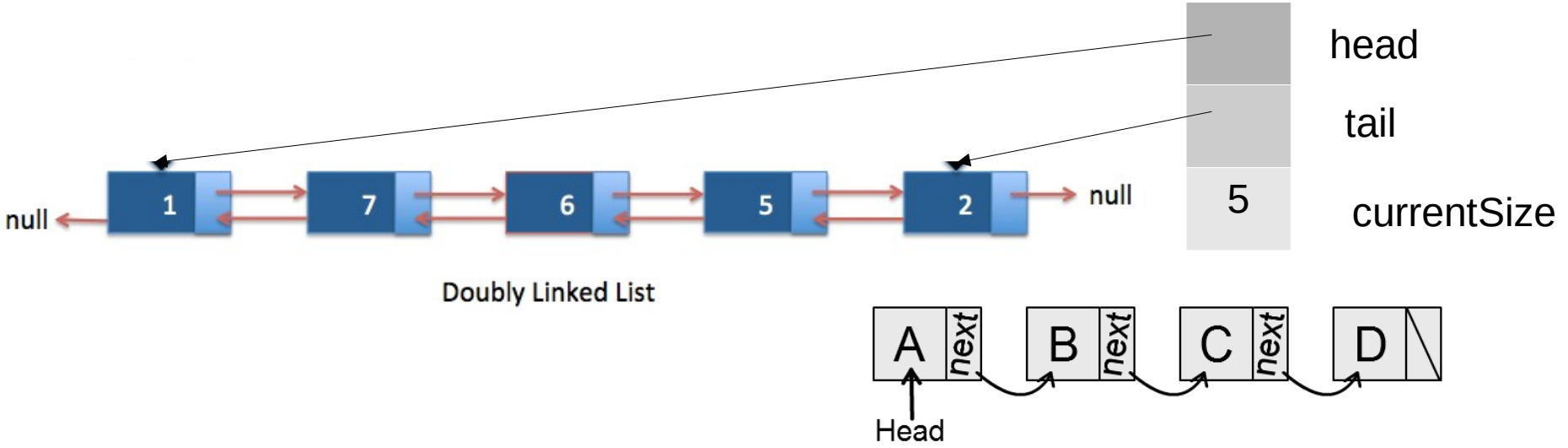
Opções de implementação

- A implementação pode ser realizada usando uma das seguintes estruturas de dados:



  - Vector ;

  - Lista ligada (simples ou dupla);



TAD **Stack<E>**

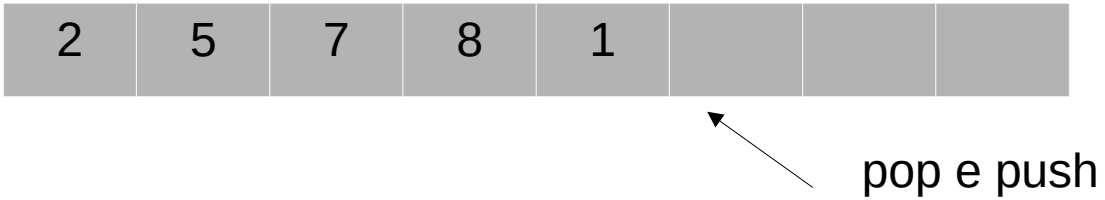
com Vector (capacidade máxima)

- Esta implementação pode ser realizada usando a classe **Array<E>.**
  - Será uma boa implementação?

TAD **Stack<E>**

com Vector (capacidade máxima)

- Esta implementação pode ser realizada usando a classe **Array<E>.**
  - Será uma boa implementação?

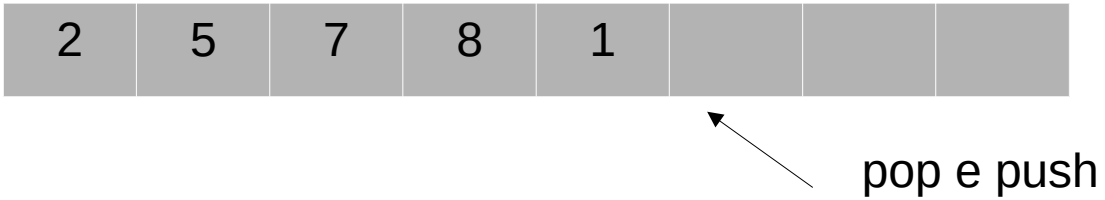


Método	Melhor caso	Caso médio	Pior Caso
addLast	O(1)	O(1)	O(1)
removeLast	O(1)	O(1)	O(1)

TAD **Stack<E>**

com Vector (capacidade máxima)

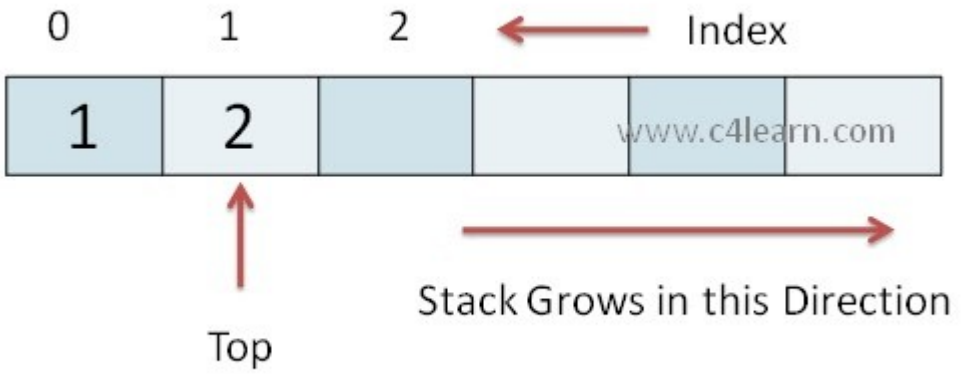
- Esta implementação pode ser realizada usando a classe **Array<E>.**
  - Será uma boa implementação?



Método	Melhor caso	Caso médio	Pior Caso
addLast	O(1)	O(1)	O(1)
removeLast	O(1)	O(1)	O(1)

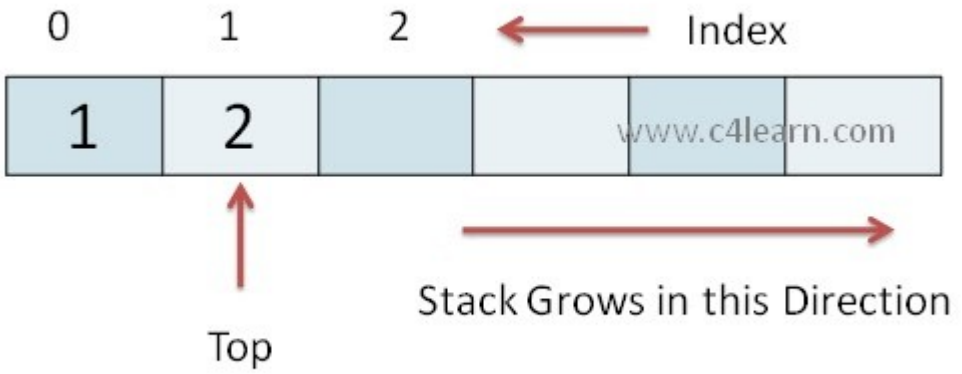
TAD Stack<E>
implementado em Vector (1)

```
package dataStructures;

public class StackInArray<E> implements Stack<E> {

    // Default capacity of the stack.
    private static final int DEFAULTCAPACITY = 1000;

    // Memory of the stack: an array.
    protected List<E> elements;

    public StackInArray(int capacity) {
        elements=new Array<E>(capacity);
    }

    public StackInArray() {
        this(DEFAULTCAPACITY);
    }
    …
}
```



TAD Stack<E>
implementado em Vector (2)

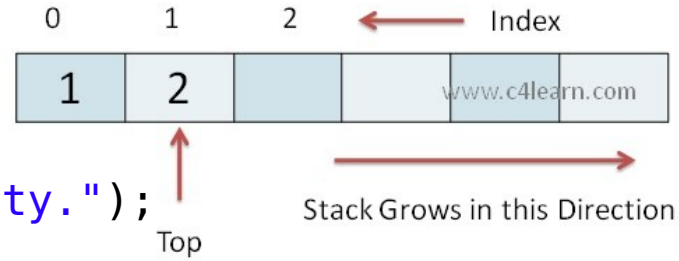
```
package dataStructures;

public class StackInArray<E> implements Stack<E> {

    @Override
    public boolean isEmpty() {
        return elements.isEmpty();
    }

    @Override
    public int size() {
        return elements.size();
    }

    ...
}
```



TAD Stack<E>

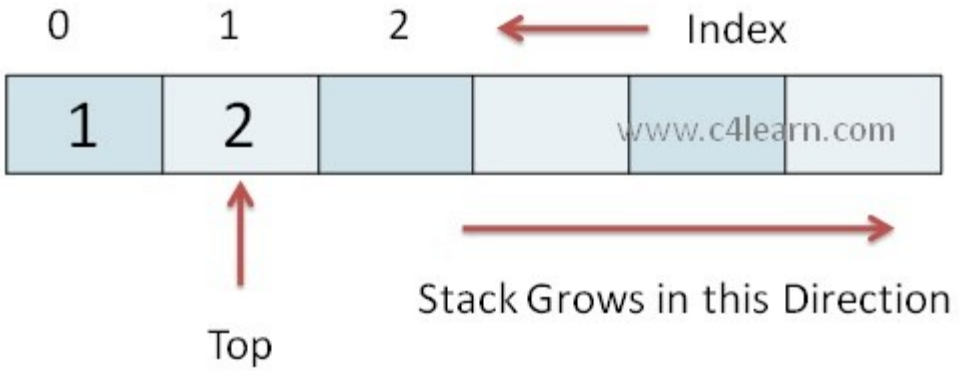
implementado em Vector (3)

```
package dataStructures;
public class StackInArray<E> implements Stack<E> {
    ...
    @Override
    public E top() throws NoElementException {
        if (isEmpty())
            throw new NoElementException("Stack is empty.");
        return elements.getLast();
    }

    @Override
    public void push(E element) throws OutOfCapacityException{
        if ( ((Array<E>) elements).isFull())
            throw new OutOfCapacityException("Stack is full.");
        elements.addLast(element);
    }

    @Override
    public E pop() throws NoElementException {
        if (isEmpty())
            throw new NoElementException("Stack is empty.");
        return elements.removeLast();
    }
}
```



TAD Stack<E>

implementado em Vector (4)



Método	Melhor caso	Caso médio	Pior caso
top	O(1)	O(1)	O(1)
push	O(1)	O(1)	O(1)
pop	O(1)	O(1)	O(1)

TAD **Stack<E>**
com Lista Ligada

- Esta implementação pode ser realizada usando as classes **SinglyLinkedList<E> ou DoublyLinkedList<E>.**
  - Qual será a melhor implementação?

Duas opções em cada uma das classes:

(A) Usar os métodos ***addLast*** e ***RemoveLast*** para implementar o ***push*** e o ***pop***, respectivamente.

(B) Usar os métodos ***addFirst*** e ***RemoveFirst*** para implementar o ***push*** e ***pop***, respectivamente

Qual a melhor opção?

TAD **Stack<E>**
com Lista Ligada

Opção/Classe	Método	Melhor Caso	Caso Médio	Pior Caso
(A) SinglyLinkedList	***push (addLast)***	O(1)	O(1)	O(1)
(A) SinglyLinkedList	**pop *(removeLast)***	O(n)	O(n)	O(n)
(B) SinglyLinkedList	***push (addFirst)***	O(1)	O(1)	O(1)
(B) SinglyLinkedList	**pop *(removeFirst)***	O(1)	O(1)	O(1)
(A) DoublyLinkedList	***push (addLast)***	O(1)	O(1)	O(1)
(A) DoublyLinkedList	**pop *(removeLast)***	O(1)	O(1)	O(1)
(B) DoublyLinkedList	***push (addFirst)***	O(1)	O(1)	O(1)
(B) DoublyLinkedList	**pop *(removeFirst)***	O(1)	O(1)	O(1)

TAD **Stack<E>**

com Lista simplesmente Ligada (1)

```
package dataStructures;

public class StackInList<E> implements Stack<E> {
    // Memory of the stack: a list.
    protected List<E> elements;

    public StackInList() {
        elements=new SinglyLinkedList<E>();
    }

    @Override
    public boolean isEmpty() {
        return elements.isEmpty();
    }

    @Override
    public int size() {
        return elements.size();
    }
    …
}
```

TAD **Stack<E>**

com Lista simplesmente Ligada (1)

```
package dataStructures;

public class StackInList<E> implements Stack<E> {

    protected List<E> elements;

    public StackInList() {
        elements=new SinglyLinkedList<E>();
    }

    @Override
    public boolean isEmpty() {
        return elements.isEmpty();
    }

    @Override
    public int size() {
        return elements.size();
    }
    …
}
```

TAD **Stack<E>**

com Lista simplesmente Ligada (2)

```
package dataStructures;

public class StackInList<E> implements Stack<E> {

    @Override
    public E top() throws NoElementException {
        if (isEmpty())
            throw new NoElementException("Stack is empty.");
        return elements.getFirst();
    }

    @Override
    public void push(E element) {
        elements.addFirst(element);
    }

    @Override
    public E pop() throws NoElementException {
        if (isEmpty())
            throw new NoElementException("Stack is empty.");
        return elements.removeFirst();
    }
}
```

Diagrama de classes
(interface **Stack**)