STACK ADT

ADT Stack

ADT Stack: sequenza lineare di elementi :

- procedure di accesso (inserzione e eliminazione) solamento al **"top"**.
  - Stack ADT segue una politica **last-in-first-out** (LIFO)

Specifica

createStack( )
// effects: Create an empty stack.

push(newElem)
// effect: Inserts newElem at the top of the stack, if there is no violation of capacit
// effect: Throws StackException if the stack is full.

isEmpty( )
// effect: Determines if a stack is empty

peek( )
// effect: Returns the top element of the stack without removing it from the stack.
// effect: Returns *null* if the stack is empty.

pop( )
// effect: Retrieves and removes the top element from the stack.
// effect: Returns *null* if the stack is empty

Specifica

createStack( )
// effects: Create an empty stack. aStack.createStack( )).isEmpty = true

push(newElem)
// effect: Inserts newElem at the top of the stack, if there is no violation of capacity.
// effect: Throws StackException if the stack is full.
/

isEmpty( )
// effect: Determines if a stack is empty

peek( )
// effect: Returns the top element of the stack without removing it from the stack.
// effect: Returns *null* if the stack is empty.

pop( )
// effect: Retrieves and removes the top element from the stack.
// effect: Returns *null* if the stack is empty

Visione assiomatica

1. (aStack.createStack( )).isEmpty = true
2. (aStack.push(Item)).isEmpty() = false
3. (aStack.createStack( )).peek( ) = null
4. (aStack.push(Item)).peek( ) = Item
5. (aStack.createStack( )).pop( ) = null
6. (aStack.push(Item)).pop( ) = Item

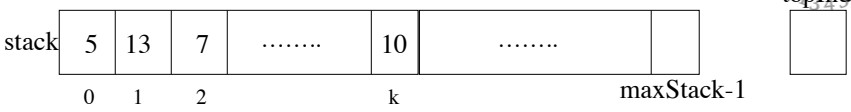
Overview

- // Overview:
- //Tipo di dato modificabili. Una struttura dati lineare dove gli elementi di tipo
- // omogeneo sono inseriti e recuperati mediante una politica LIFO.
- // Typical Element: **stack[e1, ..., top_element]**

SPECIFICA VIA JAVA INTERFACE

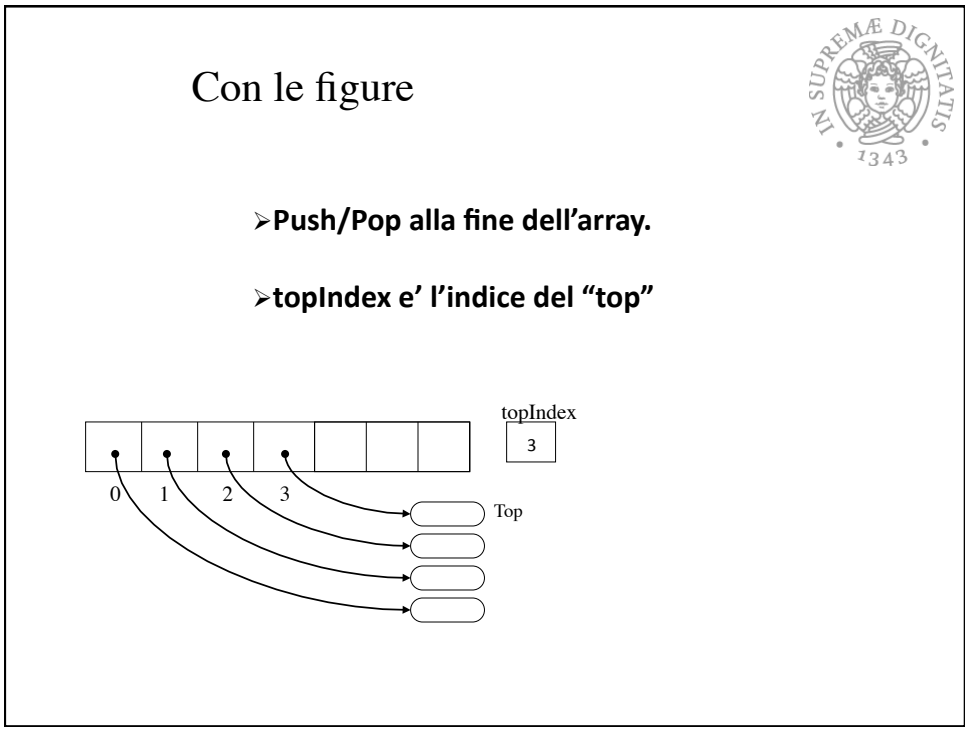
```
public interface Stack<T>{
    public boolean isEmpty();
    //Require: none
    //Effect: Returns true if the stack is empty, otherwise returns false.
    public void push(T elem) throws StackException;
    //Require: elem is the new item to be added
    //Effect: Inserts elem at the top of the stack, if there is no violation of capacity.
    //Effect: Throws StackException if the stack is full.
    public T peek()
    //Require: none
    //Effect: Returns the top element of the stack without removing it from the stack.
    //Effecr: Returns null if the stack is empty.

    public T pop()
    //Require: none
    //Effect: Retrieves and removes the top element from the stack.
    //Effect: Returns null if the stack is empty.
```

Implementazione: array-based



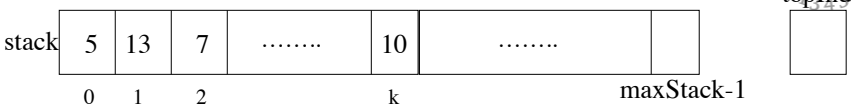

```
public class StackArrayBased<T> implements Stack<T>{
      private final int maxStack = 50;
      private T[ ] stack;
      private int topIndex;
      public StackArrayBased( ){
        stack=(T[ ])new Object[maxStack];
        topIndex = -1;
      }
      public boolean isEmpty( ){
        return (topIndex < 0);}
}
```

Con le figure



- **Push/Pop alla fine dell'array.**
- **topIndex e' l'indice del "top"**

Implementazione: array-based




```
public class StackArrayBased<T> implements Stack<T>{
      private final int maxStack = 50;
      private T[ ] stack;
      private int topIndex;
      public StackArrayBased( ){
        stack=(T[ ])new Object[maxStack];
        topIndex = -1;
      }
      public boolean isEmpty( ){
        return (topIndex < 0);}
}
```

Funzione di astrazione


```
  private final int maxStack = 50;
  private T[ ] stack;
  private int topIndex;
```

```
Alpha(c.stack) = stack[e1,… ek] &
  c.stack[top_index] = e_k =
  top_element
```

Invariante di rappresentazione

- I(c) = top_index > = 0 &
  \forall index, 0, top_index-1 c.stack[i].type= T

Che ruolo gioca

- La dimensione massima dello stack?
- Cosa si deve cambiare?

```
public void push(T elem) throws
StackException{
    topIndex++;
    if (topIndex >= maxStack)
        throw new StackException("stack
full");
    stack[topIndex] = item;
}
```

dynamic expansion

doubleArray();

```
public T peek( ){
  T top = null;
  if ( !isEmpty())
    top =
stack[topIndex];
  return top;
}
```

peekFirst()

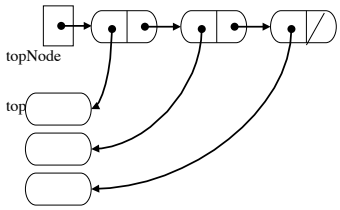
Java 6 API

```
public T pop( ){
    T top = null;
    if (!isEmpty()){
        top = stack[topIndex];
        stack[topIndex] = null;
        topIndex--;}
    return top
}
```

removeFirst( )

Java 6 API

Implementazione list based

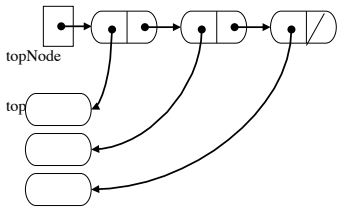



```
class Node<T>{
  private T element;
  private Node<T> next;
  ……
}
```

```
class LinkedBasedStack<T>{
    private Node<T> topNode;
      ……
}
```

Implementation of Stack<T> methods

Implementazione list based



Alpha(c) = [c.topNode.getElem] @
alpha(c.topNode.next)

I(c) = c,topNode= null & c.isEmpty = true or
!c.topNode = null & c.isEmpty = False

```
class Node<T>{
  private T element;
  private Node<T> next;
  ……
}
```

```
class LinkedBasedStack<T>{
    private Node<T> topNode;
      ……
}
```

Implementation of Stack<T> methods

```
public void push(T newElem){
      topNode = new Node<T>(newElem, topNode);
}
public T peek( ) {
       T top =  null;
          if (!isEmpty) top = topNode.getElem( );
       return top;
}
public T pop( ) {
      T top =  null;
         if (!isEmpty) {
             top = topNode.getElem( );
             topNode = topNode.getNext();
      }
      return top;
}
```

JAVA Collection

Java interface `Deque<E>`

Stack Methods	Deque Methods
push(elem)	addFirst(elem)
pop()	removeFirst(elem)
peek()	peekFirst(elem)

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

ADT Queue

ADT Queue sequenza linerare di elementi omeogenei:

Procedure di accesso permettono l'inserimento solo nella coda mentre l'eliminazione avviene in test.
Typical element: Queue[back, e2, … , ek, front]

- Una coda segue una politica di tipo **first-in-first-out** (FIFO).

Specifica

createQueue
// Effect: Create an empty queue

isEmpty( )
// Effect: Determines if a queue is empty

getFront( )
// effect: Returns, but does not remove, the head of the queue.
// effect: Throws QueueException if the queue is empty.

enqueue(newElem)
// effect: Inserts newElem at the back of the queue, if there is no violation of
// effect: capacity. Throws QueueException if the queue is full.

dequeue( )
// effect: Retrieves and removes the head of the queue. Throws QueueException
// effect: if the queue is empty.

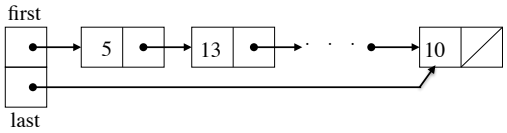
Visione Assiomatica

1. (aQueue.createQueue( )).isEmpty = true
2. (aQueue.enqueue(Elem)).isEmpty() = false
3. (aQueue.createQueue( )).getFront( ) = error
4. pre: aQueue.isEmpty() = true:
   (aQueue.enqueue(Elem)).getFront( ) = Elem
5. (aQueue.createQueue( )).dequeue( ) = error
6. pre: aQueue.isEmpty( )=false:
   (aQueue.enqueue(Elem)).dequeue( )=(aQueue. dequeue( )). enqueue(Elem)

Java Interface

```
public interface Queue<T>{

    public boolean isEmpty();
    //Pre: none
    //Post: Returns true if the queue is empty, otherwise returns false.

    public void enqueue(T elem) throws QueueException;
    //Pre: item is the new item to be added
    //Post: If insertion is successful, item is at the end of the queue.
    //Post: Throws QueueException if the item cannot be added to the queue.

    public T getFront( ) throws QueueException;
    //Pre: none
    //Post: If queue is not empty, the item at the front of a queue is returned, and the queue
    //Post: is left unchanged. Throws QueueException if the queue is empty.

    public T dequeue( ) throws QueueException;
    //Pre: none
    //Post: If queue is not empty, the item at the front of the queue is retrieved and removed
    //Post:from the queue. Throws QueueException if the queue is empty.

}//end of interface
```

Implmenentazione: List based




```
class Node<T>{
  private T element;
  private Node<T> next;
  ……
}
```

```
class LinkedBasedQueue<T>{
    private Node<T> first;
    private Node<T> last;
      ……
}
```