|  |
| --- |
| TAD Stack <T> |
| Stack = (Node<T>1, Node<T>2, Node<T>3, ..., Node<T>n), n >= 0, Node<T>1 = Top V Node<T>2 to Node<T>n = elements |
| {Inv.: - ∀x, x ∈ Stack, x ≠ NULL   * the elements are ordered in a last-in, first-out (LIFO) fashion, Stack = (Node<T>1, Node<T>2, Node<T>3, ..., Node<T>Tx), Node<t>X = Added element. * Removed Element = E1}. |
| Primitive Operations:   * PushStack: Node<T> x Stack → Stack (Modifier). * PopStack: Stack → Node<T> (Modifier). * TopStack: Stack → Node<T> (Observer). * IsEmptyStack: Stack → Boolean (Observer). * SizeStack: Stack → Integer (Observer). |

|  |
| --- |
| PushStack(Node<T>)  “This operation adds an element to the top of the stack, modifying the stack.”  {pre: TRUE}  {post: The element is added to the top of the stack. Element = En.} |

|  |
| --- |
| PopStack()  “This operation removes and returns the element at the top of the stack, modifying the stack.”  {pre: Stack.size ≠ 0 v Stack.isEmptyStack}  {post: Stack = ( Node<T>2, Node<T>3, ..., <Node>n), New stack without first element.} |

|  |
| --- |
| TopStack()  “This operation returns the element at the top of the stack without removing it.”  {pre: Stack.size ≠ 0 v Stack.isEmptyStack}  {post: returns Element = Nod<T>1, ∧ Stack = (Node<T>1, Node<T>2, Node<T>3, ..., Node<T>n),} |

|  |
| --- |
| IsEmptyStack()  “This operation checks whether the stack is empty and returns a boolean value.”  {pre: No specific preconditions}  {post: Stack = (Node<T>1, Node<T>2, Node<T>3, ..., Node<T>n),  True if Node<T>n == NULL,  False if Node<T>n ≠ NULL} |

|  |
| --- |
| SizeStack()  “This operation returns the number of elements currently in the stack.”  {pre: No specific preconditions}  {post: Stack = (Node<T>1, Node<T>2, Node<T>3, ..., En),  Returns Int = n) |