|  |
| --- |
| TAD Heap <T> (Max Heap) |
| Heap<T> = ArrayList = {Node<T>1, Node<T>2, Node<T>3, ..., Node<T>n), n >= 0} Node<T>.data ≠ null, Node<T>1 = Parent, ArrayList[1] to ArrayList[n] = Sons |
| {Inv.: - Node<T>1.data> Node<T>2.data>…. Node<T>n.data }. |
| Primitive Operations:   * Insert: Heap<T> x Node<T> → Heap. (Modifier.) * ExtractMin: Heap<T> → Node<T>(Modifier.) * Heapify: Heap<T> → Heap<T> (Modifier. ) * IsEmpty: Heap<T> → Boolean (Analyzer.) * Heap: → Heap<T> (Constructor.) |

|  |
| --- |
| Insert(Node<T>)  “Add a node to the heap while maintaining the heap property.”  {pre: Node<T>.data ≠ null }  { post: Heap<T>.size = Heap<T>.size + 1 && Heap<T> = newHeap<T>} |

|  |
| --- |
| ExtractMin()  "Remove and return the minimum node from the heap while maintaining the heap property".  {pre: Node<T>.data ≠ null && Heap<T> ≠ null }  { post: result = Heap<T>.min && Heap<T>.size = Heap<T>.size\_old – 1 } |

|  |
| --- |
| Heapify()  "Convert an array or collection into a valid heap".  {pre: Node<T>.data ≠ null && Heap<T> ≠ null }  {post: Heap<T> = newArrangedHeap<T>} |

|  |
| --- |
| IsEmpty()  "Check if the heap is empty.”  {pre: A Heap<T> must be instanciated.}  {post: post: result = (heap.size == 0) } |