|  |  |
| --- | --- |
| NAME | QUEUE |
| REPRESENTATION | Java Queue  Queue = <<e1, e2,…,en>,front,back> |
| INVARIANT | Size(Queue) = n front = e1 back = en 0 >= n |
| OPERATIONS | Queue - 🡪 Queue  Offer Queue x Element -> Queue  Poll Queue 🡪 Element  Peek Queue 🡪 Element  Size 🡪 Integer  IsEmpty🡪boolean  Clear 🡪 void |

Constructor operation

|  |
| --- |
| Queue - 🡪 Queue  “Builds an empty queue”  {Pre: -}  {Pos: Queue q =} |

Modifier operation

|  |
| --- |
| Offer Queue x Element -> Queue  “Insert a new Element e in the Queue”  {Pre: Queue q = <e1, e2,…,en> and element e or q = ø and element e}  {Pos: Queue q = <e1, e2,…,en, e> or q = <e> } |

|  |
| --- |
| Poll Queue 🡪 Element  “Extracts the element in Queue q’s front”  {Pre: ø i.e. q = <e1, e2,…,en> }  {Pos: Queue q = <e2,e3,…,en-1> and element e1 } |

|  |
| --- |
| Peek Queue 🡪 Element  “Recovers the value of the element on the front of the queue.”  {Pre: Queue q != i.e q = <e1, e2,…,en> }  {Pos: Element e1} |

|  |
| --- |
| Size 🡪 integer“Get the QUEUE size” {Pre: Queue q = or q = <e1, e2,…,en> }  {Pos: 0 or n} |

|  |
| --- |
| IsEmpty 🡪 boolean“Determines if the Queue q is empty or not” {Pre: Queue q}  {Pos: true if queue is empty, false if is not} |

Destructor operation

|  |
| --- |
| Clear“Destroys queue q freeing memory” {Pre: Queue q}  {Pos: -} |

|  |  |
| --- | --- |
| NAME | HASH TABLE |
| REPRESENTATION | HashTable in Java Example | Java HashTable Tutorial |
| INVARIANT | Inv {All keys cannot be null,  Every value will have a key} |
| OPERATIONS | HashTable 🡪 HashTableHashFunction: Key Search: Key 🡪 Value  Insert: Key, Value  Delete: Key |

Constructor operation

|  |
| --- |
| HashTable 🡪 HashTable  “Creates a new hashTable”  {Pre: true}  {Pos: An empty hashTable} |

Modifiers operation

|  |
| --- |
| HashFunction: Key  “Convert the key to an integer”  {Pre: True}  {Pos: An integer that indicates the object position in the array} |

|  |
| --- |
| Search: Key 🡪 Value  “Get the value searched”  {Pre: ø}  {Pos: Returns the element found} |

|  |
| --- |
| Insert: Key, Value  “Add a new element in the hash table”  {Pre: Key && Value ≠ null}  {Pos: Insert the element in the hash table} |

|  |
| --- |
| Delete: Key  “Delete an element with a key”  {Pre: Key ≠ null}  {Pos: Erase the found element} |

|  |  |
| --- | --- |
| NAME | STACK |
| REPRESENTATION | Stack Class in Java Explained with Examples | CodeAhoy  Stack = <<e1, e2,…,en>,top > |
| INVARIANT | n >= 0 Size = n top = en |
| OPERATIONS | Stack - 🡪 Stack  Pop Stack 🡪 Element  Top Stack 🡪 Element  Push Stack x Element -> Stack  isEmpty Stack 🡪 Boolean  Size 🡪 integer  Clear |

Constructor operation

|  |  |
| --- | --- |
| Stack - 🡪 Stack  “Builds an empty stack”  {Pre: -}  {Pos: Stack s = } |  |

Modifiers operations

|  |
| --- |
| Push Stack x Element -> Stack  “Adds the new element e to stack s”  {Pre: Stack s = and element e or s = <e1, e2,…,en> and element e}  {Pos: Stack s = s = <e1, e2,…,en, e> or s = <e> } |

|  |
| --- |
| isEmpty Stack 🡪 Boolean  “Determines if the stack s is empty or not”  {Pre: Stack s}  {Pos: true if s = False if s != } |

|  |
| --- |
| Top Stack 🡪 Element  “Recovers the value of the element on the top of the stack”  {Pre: : Stack s != i.e s = <e1, e2,…,en> }  {Pos: Element en} |

|  |
| --- |
| Pop Stack 🡪 Element  “Extracts from the stack s, the most recently inserted element, and recover that element”  {Pre: Stack s != i.e s = <e1, e2,…,en> }  {Pos: Stack s = <e1, e2,…,en-1> , Element en} |

|  |
| --- |
| Size 🡪 integer  “Get the size of the Stack”  {Pre: Stack s = <e1, e2,…,en> or s = }  {Pos: n or 0} |

Destructor operation

|  |
| --- |
| Clear “Destroys stack s freeing memory” {Pre: Stack s}  {Pos: -} |

|  |  |
| --- | --- |
| NAME | HEAP |
| REPRESENTATION | Heap Data Structure - GeeksforGeeks |
| INVARIANT | Size >= 0 |
| OPERATIONS | Heap🡪 Heap  buildMaxHeap  maxHeapify  exist 🡪 Boolean |

Constructor operation

|  |
| --- |
| Heap - 🡪 Heap  “Builds an empty heap”  {Pre: -}  {Pos: Heap h = } |

Modifier operations

|  |
| --- |
| Maxheapify  “moves the element to the correct position”  {pre: Heap h}  {pos: Heap h} |
| BuildMaxheap  “Moves all the elements to the correct position to start the heapsort”  {pre: Heap h}  {pos: Heap h in descendent order} |

|  |
| --- |
| HeapSort  “Order all the elements”  {pre: Heap h with buildMaxheap}  {pos: heap h in ascendant order} |

|  |
| --- |
| Swap  “swap two elements in the heap h”  {pre: heap h}  {pos: heap h} |

|  |
| --- |
| Exist int i  “determinates if that position exist”  {pre: heap h}  {pos: true if ei exist, false if not} |