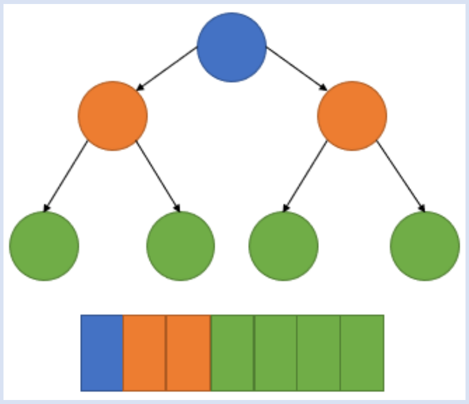
TAD **Heap <T,V>**



{inv: The values of each node are less than or equal to the values of its children**}**

<u>Primitive operations:</u>

• father:	int	->int
• left:	int	->int
• right:	int	->int
• addHeapNode:	T,V	->boolean
• heapify:	int	->HeapNode<T,V>[]
• buildHeap:		
• getArraySize:		->int
• getArray:		->HeapNode<T,V>[]

father
"return the floor of n/2 of the father node"
{pre: the father node must exist}
{post: floor of n/2}
Analyzer

left
"return the floor of n/2 of the left node"
{pre: the left node must exist}
{post: floor of n/2}
Analyzer

right
"return the floor of n/2 of the right node"
{pre: the right node must exist}
{post: floor of n/2}
Analyzer

addHeapNode
"add a node to the heap"
{pre: receives the parameters T y V}
{post: node has been added}
Modifier

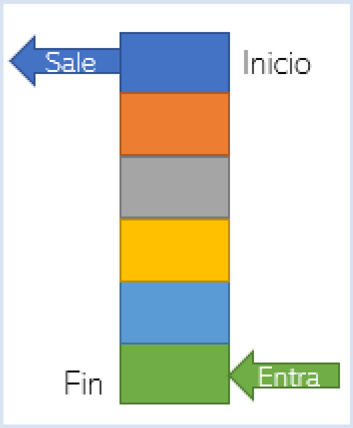
heapify
"the node with the highest K becomes the father node"
{pre: receives the position of the largest K}
{post: the heapify has been completed}
Builder

buildHeap
"build a heap"
{pre: TRUE}
{post: Heap has been created}
Builder

getArraySize
"return the array size of the heap"
{pre: TRUE}
{post: Heap size}
Analyzer

getArray
"return the array of the heap"
{pre: TRUE}
{post: Array of the heap}
Analyzer

TAD Queue <T>



{inv: First in, first out}

Primitive operations:

• front:		->QueueNode<T>
• enqueue:	T	
• isEmpty:		->boolean
• dequeue:		->QueueNode<T>

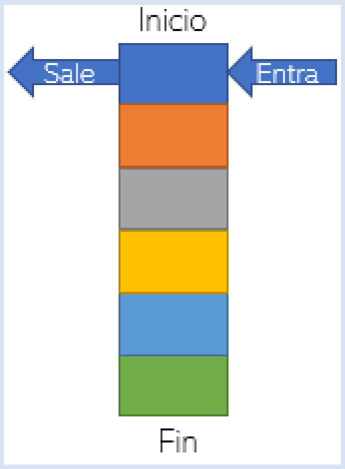
front
"return the node of queue that is in the first position"
{pre: TRUE}
{post: Queue node in the first position}
Analyzer

enqueue
"add an element T to the end of the Queue"
{pre: receives a T element}
{post: The element has been added}
Builder

isEmpty
"check that the queue is empy"
{pre: TRUE}
{post: boolean indicating whether the queue is empty}
Analyzer

dequeue
"remove the first element of the queue"
{pre: TRUE}
{post: return element T removed}
Modifier

TAD Stack<V>



{inv: *Last in, first out*}

Primitive operations:

- isEmpty: ->boolean
- push: V
- top: ->StackNode<V>
- pop: ->StackNode<V>

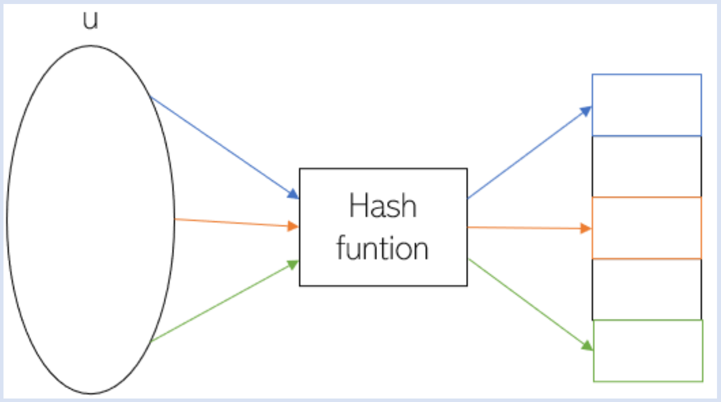
isEmpty
"check that the stack is empy"
{pre: TRUE}
{post: boolean indicating whether the stack is empty}
Analyzer

push
"add a value V to the start of the stack"
{pre: receives the value V}
{post: The value has been added}
Modifier

top
"return the node of stack that is in the first position
{pre: TRUE}
{post: Stack node in the first position}
Modifier

pop
“remove the first value of the Stack”
{pre: TRUE}
{post: return value V removed}
Modifier

TAD Hash <K,V>



{inv: $Efficient\ search. Allows\ access\ to\ stored\ items\ from\ a\ generated\ key$}

Primitive operations:

• add:	K,V	
• getValue:	K	->V
• remove:	K	->boolean
• contains:	K	->boolean
• getSize:		->long
• hashFuntion:	K	->int

add
"add hash node"
{pre: receives a key K and a value V}
{post: hash node has been added}
Modifier

getValue
"return the value V of a give key K"
{pre: receives a key K}
{post: return V corresponding to K}
Analyzer

remove
"remove the value of a given key"
{pre: receives a key K}
{post: boolean indicating that V corresponding to K has been removed}
Modifier

contains
“verifies the exisrence of the given key K”
{pre: receives a key K}
{post: boolean indicating if the key K exists}
Analyzer

getSize
“return the size of the hash table”
{pre: TRUE}
{post: Hash table size}
Analyzer

hashFuntion
“optimize data search in the hash table”
{pre: receives a key K}
{post: position to which the value was added}
Analyzer