|  |
| --- |
| **TAD Colas de prioridad** |
|  |
| {inv: (elements={e1, e2,..en}∧ priorities={p1,p2,...pn} ⩝ pi ⩾pj ⟷ ei ﹥prioridad que ej →1≤ i ∧ j ≤ n) }  {inv: (elements:{A,B ,C, D} ∧ priorities={3, 6, ,5, 1}= 3≤5≤,6≤1)} |
| * PQueue() →PQueue * insert() elemento genericoT → PQueue * delete() PQueue → element generic T * swap() índice x índice → PQueue * isEmpty() PQueue →empty or not PQueue * print() PQueue → PQueue |
| PQueue()(Constructor)  “Crea una cola de prioridad vacía”  {pre: PQueue !=null}  {post PQueue pq=0:} |
| insert(<T>)(Modificador)  “Inserta un nuevo elemento <T> al final de la cola”  {pre:PQueue pq= (e1,e2,...en) ∧ element e ∧ q=0 }  {post:PQueue pq=(e1,e2,...en)} |
| delete(PQueue)(modificador)  “Elimina el elemento de mayor prioridad de la cola de prioridad y lo devuelve.”  {pre:PQueue pq ≠0 ∨ pq=(e1,e2,...en)}  {post:PQueue pq= (e1,e2,...en-1)} |
| swap(i , j)(modificador)  “Intercambia los elementos en las posiciones **i** y **j**”  {pre: PQueue pq=(e1,e2,...en)}  {post: PQueue()} |
| isEmpty(PQueue)(analizador)  “Define si la cola está vacía”  {pre: PQueue pq}  {post:If pq ≠ 0→ FALSE; else (es decir, pq=0)→ TRUE } |

|  |
| --- |
| **TAD Colas** |
| Queue=[(e1,e2,...en), front, back] |
| {inv: 0≤n ∧ size(Queue)=n ∧ front= e1 ∧ back=en} |
| * Queue() → Queue * enqueue() Queue→e → Queue * dequeue() Queue → e * peek () Queue → e1 * isEmpty() Queue → TRUE ∨ FALSE |
| Queue() (Constructor)  “Crea una cola vacía”  {pre: Queue !=null}  {post: Queue=0} |
| enqueue(Queue,e) (Modificador)  “Inserta un elemento <e> al final de la cola <q>”  {pre: Queue q= (e1,e2,...en) ∧ e ∨ q=0 ∧ e}  {post: Queue q= (e1,e2,...en+1)} |
| dequeue(Queue)(Modificador)  “Extrae un elemento <e> del frente de la cola <q>”  {pre: Queue q=0 ∨ q=(e1,e2,...en)}  {post:Queue q=(e2,e3...en-1)∧ e1 } |
| peek (Queue)(Analizador)  “Muestra el valor de <e> que esté al frente de la cola <q>”  {pre: Queue q=0 ∨ q=(e1, e2,...en)}  {post: e1} |
| isEmpty(Queue)(Analizador)  “Determina si la cola <q> está vacía o no”  {pre: Queue q}  {post: TRUE if q=0, FALSE if q≠0} |

|  |
| --- |
| **HashMap** |
|  |
| {inv: la key no se repite en ningún otro elemento}  {inv: si hay colisión, se almacenara en una linkedlist} |
| * HashTable () lenght → HashTable * Insert() HashTable x key genéric x value generic → HashTable * Get() HashTable x key generic → element generic V * Remove() HashTable x key generic → HashTable * Print() HashTable → text * Hash() HashTable x generic key → index * isEmpty() HashTable → TRUE ∨ FALSE |
| HashTable()(Constructor)  “Crea una hashtable vacio”  {pre: HashTable !=null}  {post HashTable pq=length:} |
| insert(HashTable ,K,V)(Modificador)  “Inserta un elemento con una clave K y un valor V”  {pre:HashTable.lenght>0 }  {post:Hashtable(v1,v2,v3….vn)} |
| get(HashTable,K)(Analizador)  “Retorna el valor asociado a una clave”  {pre:HashTable.lenght>0 y K exist}  {post:V v null} |
| Remove (HashTable,K)(modificador)  “Elimina el elemento asociado a una clave”  {pre:HashTable.lenght>0 y K exist}  {post: :HashTable } |
| Print(HashTable)(Analizador)  “Retorna una cadena de caracteres con los valores”  {pre:HashTable.lenght>0 y hasTable ¡=null}  {post:text } |
| Hash(HashTable,K)(Analizadora)  “Calcula el índice de la tabla hash para una clave”  {pre: }  {post: postive index } |
| isEmpty(HashTable)(Analizador)  “Determina si la Hashtable está vacía o no”  {pre: Hashtable exist}  {post: TRUE if Hashtable =0, FALSE if Hashtable ≠0} |