**TEST CASES HASHTABLE**

| **Name** | **Class** | **Scenery** |
| --- | --- | --- |
| setUp1 | TestHashtable | A task with{  name=Task1  description=Hard  endDate=4-apr-2023  Priority=priority  } |
| setUp3 | TestHashtable | hash= new hashtable |
| setUp1 | DoubleLinkedList | A new list→listdob= new DoubleLinkedList |
| setUp2 | TestDoubleLinkedList | The list with elements added in the front{  listDob.addFirst(6);  listDob.addFirst(8);  listDob.addFirst(15);  } |
| setUp3 | TestDoubleLinkedList | The list with elements added in the back{  listDob.addLast(6);  listDob.addLast(8);  listDob.addLast(15);  } |
| setUp1 | TestStack | Declaration of a new stack  stack= new Stack<Action> |
| setUp2 | TestStack | Stack with one Action{  type=modify  task=null  originalTask=null  } |
| setUp3 | TestStack | Stack with two Action{  task=null  originalTask=null  type=add  task=null  originalTask=null  }type=modify |
| setUp1 | TestQueue | Declaration of a new Queue  tail= new Queue |
| setUp3 | TestQueue | The tail with two element task{  name=Task1  description= hard  key =50  finalDate=2023- apr-01  name=Task2  description= easy  key =32  finalDate=2023- may-02  } |
| setUp2 | TestQueue | The tail with one element task{  name=Task1  description= hard  key =50  finalDate=2023- apr-01  } |
| setUp1 | TestMaxHeap | Inicializa new MaxHeap  priority= new MaxHeap<Task>(20); |
| setUp2 | TestMaxHeap | A MaxHeap with one element  {  name= Task1  description= nada  key=50  endDate=2024-may-2  } |
| setUp3 | TestMaxHeap | A MaxHeap with two element  {  name= Task1  description= null  key=50  endDate=2023-may-27  name= Task2  description= null  key=23  endDate=2024-apr-2  } |
| setUp4 | TestMaxHeap | A MaxHeap with 4 element  {  name= 1  description= null  key=50  endDate=2023-may-27  name= 2  description= null  key=23  endDate=2024-apr-2  name= 3  description= null  key=15  endDate=2024-may-2  name= 4  description= null  key=31  endDate=2024-jun-2  } |

**TEST CASES FOR HASHTABLE**

| **Objective of the test: Test if a task is correctly added to the hashtable** | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenery** | **Entrance value** | **Result** |
| Hastable | insert | setUp1 | A task with{  name=Task1  description=Hard  endDate=4-apr-2023  Priority=priority  } | The element is correctly added at the hash table |

| **Objective of the test: Test if a task is correctly deleted to the hashtable** | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenery** | **Entrance value** | **Result** |
| Hastable | delete | setUp1 | A task with{  name=Task1  description=Hard  endDate=4-apr-2023  Priority=priority  } | The element is correctly deleted in the hash table |

| **Objective of the test: Test if a task can be restored after deleting it from the hashtable** | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenery** | **Entrance value** | **Result** |
| Hastable | restore | setUp3 | hash= new Hashtable | the element is removed from the hashtable, then the action is undone and the element is reintegrated into the hash. |

| **Objective of the test: Test if the method search find an element given an index** | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenery** | **Entrance value** | **Result** |
| Hastable | search | setUp1 | A task with{  name=Task1  description=Hard  endDate=4-apr-2023  Priority=priority  } | The method returns correctly the element that previously was searched given an index. |

**TEST CASES STACK**

| **Objective of the test: Test if an object is correctly added to the stack** | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenery** | **Entrance value** | **Result** |
| Stack | push | setUp1 | stack= new Stack | The object(Action) is correctly added to the stack |

| **Objective of the test: Test if an object is correctly deleted of the stack** | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenery** | **Entrance value** | **Result** |
| Stack | pop | setUp2 | Stack with one Action{  type=modify  task=null  originalTask=null  } | The object(Action) is correctly deleted of the stack |

| **Objective of the test: Test if the method returns the element at the top of the stack** | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenery** | **Entrance value** | **Result** |
| Stack | top | setUp3 | Stack with two Action{  task=null  originalTask=null  type=add  task=null  originalTask=null  }type=modify | The method returns the action at the top of the stack |

| **Objective of the test: Test if the method calculates correctly the size of the stack after making any operation, in this case pop** | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenery** | **Entrance value** | **Result** |
| Stack | size | setUp3 | Stack with two Action{  task=null  originalTask=null  type=add  task=null  originalTask=null  }type=modify | The method returns the actual value of the stack size after deleting an element of the stack |

| **Objective of the test: Test if the stack is empty** | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenery** | **Entrance value** | **Result** |
| Stack | empty | setUp1 | stack= new Stack | The method returns true if the size of the stack is zero |

**TEST CASES QUEUE**

| **Objective of the test: Test if an element is correctly added to the Queue** | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenery** | **Entrance value** | **Result** |
| Queue | enQueue | setUp1 | tail= new Queue | A new Task is created and correctly added to the queue |

| **Objective of the test: Test if an element is correctly deleted of the Queue** | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenery** | **Entrance value** | **Result** |
| Queue | deQueue | setUp2 | {  name=Task1  description= hard  key =50  finalDate=2023- apr-01  } | the Task previously added is deleted of the queue and the size decrease |

| **Objective of the test: Test if the method returns the element in the front of the Queue** | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenery** | **Entrance value** | **Result** |
| Queue | front | setUp3 | {  name=Task1  description= hard  key =50  finalDate=2023- apr-01  name=Task2  description= easy  key =32  finalDate=2023- may-02  } | The method returns the actual front of the queue |

| **Objective of the test: Test if the queue is empty** | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenery** | **Entrance value** | **Result** |
| Queue | empty | setUp1 | tail= new Queue | The method return true if the queue does not have tasks stored |

| **Objective of the test: Test if the method return the actual size of the queue** | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenery** | **Entrance value** | **Result** |
| Queue | size | setUp3 | {  name=Task1  description= hard  key =50  finalDate=2023- apr-01  name=Task2  description= easy  key =32  finalDate=2023- may-02  } | The method return the actual size of the queue after operations |

**TEST CASES MAXHEAP**

|  | | | | |
| --- | --- | --- | --- | --- |

| **Objective of the test: Verify if the heapify order correctly the elements stored** | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenery** | **Entrance value** | **Result** |
| MaxHeap | maxHeapify | setUp4 | {  name= 1  description= null  key=50  endDate=2023-may-27  name= 2  description= null  key=23  endDate=2024-apr-2  name= 3  description= null  key=15  endDate=2024-may-2  name= 4  description= null  key=31  endDate=2024-jun-2  } | after inserting all was in disorder, the expected result should be that the most priority task is at the maximum position of the priority Structure |

| **Objective of the test: Test if is an element (Task) is correctly inserted in the MaxHeap** | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenery** | **Entrance value** | **Result** |
| MaxHeap | insert | setUp2 | {  name= Task1  description= nada  key=50  endDate=2024-may-2  } | The Task is correctly added to the priority MaxHeap |

| **Objective of the test: Test if a element is correctly extracted of the MaxHeap** | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenery** | **Entrance value** | **Result** |
| MaxHeap | extractMax | setUp3 | {  name= Task1  description= null  key=50  endDate=2023-may-27  name= Task2  description= null  key=23  endDate=2024-apr-2  } | The Task is correctly deleted and extracted of the priority MaxHeap |

| **Objective of the test: Test if a element is correctly deleted of the MaxHeap** | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenery** | **Entrance value** | **Result** |
| MaxHeap | remove | setUp3 | {  name= Task1  description= null  key=50  endDate=2023-may-27  name= Task2  description= null  key=23  endDate=2024-apr-2  } | The Task is correctly deleted of the priority MaxHeap |

| **Objective of the test: Test if given an object we can find the index of that element in the MaxHeAP** | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenery** | **Entrance value** | **Result** |
| MaxHeap | getIndexForAnObject | setUp2 | {  name= Task1  description= null  key=50  endDate=2023-may-27  } | The method returns the index of the object specified in the parameter |

| **Objective of the test: Test if the MaxHeap is empty** | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenery** | **Entrance value** | **Result** |
| MaxHeap | isEmpty | setUp | -priority=new MaxHeap | The method returns true indicating that that the MaxHeap is empty |

| **Objective of the test: Test if the method is showing the correct information** | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenery** | **Entrance value** | **Result** |
| MaxHeap | print | setUp2 | {  name= Task1  description= null  key=50  endDate=2023-may-27  } | The method prints correctly the information |

**TEST CASES DOUBLE LINKED LIST**

| **Objective of the test: Test if the method add first is working properly** | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenery** | **Entrance value** | **Result** |
| DoubleLinkedList | addFirst | setUp2 | listDob.addFirst(6);  listDob.addFirst(8);  listDob.addFirst(15); | The elements are inserted one by one at the beginning of the list |

| **Objective of the test: Test if the method add first is working properly** | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenery** | **Entrance value** | **Result** |
| DoubleLinkedList | addFirst | setUp2 | listDob.addFirst(6);  listDob.addFirst(8);  listDob.addFirst(15); | The elements are inserted one by one at the beginning of the list |

| **Objective of the test: Test if the method add last is working properly** | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenery** | **Entrance value** | **Result** |
| DoubleLinkedList | addLast | setUp3 | listDob.addLast(6);  listDob.addLast(8);  listDob.addLast(15); | The elements are inserted one by one at the back of the list |

| **Objective of the test: Test if the find node method search correctly a node given an index** | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenery** | **Entrance value** | **Result** |
| DoubleLinkedList | findNode | setUp3 | listDob.addLast(6);  listDob.addLast(8);  listDob.addLast(15); | The method returns the node that correspond to the given index |

| **Objective of the test: Test if the method deleteFirst deletes correctly the first node of the double linked list** | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenery** | **Entrance value** | **Result** |
| DoubleLinkedList | deleteFirst | setUp3 | listDob.addLast(6);  listDob.addLast(8);  listDob.addLast(15); | The method deletes correctly the element at the beginning of the double linked list |

| **Objective of the test: Test if the method delete, eliminates correctly a node of the double linked list indicated by an index** | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenery** | **Entrance value** | **Result** |
| DoubleLinkedList | delete | setUp3 | listDob.addLast(6);  listDob.addLast(8);  listDob.addLast(15); | The method deletes correctly the element specified by an index as a parameter in the double linked list |

| **Objective of the test: Test if the method deleteLast deletes correctly the last node of the double linked list** | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenery** | **Entrance value** | **Result** |
| DoubleLinkedList | deleteLast | setUp3 | listDob.addLast(6);  listDob.addLast(8);  listDob.addLast(15); | The method deletes correctly the element at the final of the double linked list |

| **Objective of the test: Test if a element is correctly modified given an index and the new content** | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenery** | **Entrance value** | **Result** |
| DoubleLinkedList | modifyContent | setUp3 | listDob.addLast(6);  listDob.addLast(8);  listDob.addLast(15); | The method replaces the previous content of the node indicated by the index for the new value |