## Hash Table

|  |  |  |
| --- | --- | --- |
| **Name** | **Class** | **Scenery** |
| hashTable\_setup1 | HashTableTest | Hash table is empty |
| hashTable\_setup2 | HashTableTest | The hash table has added values. |
| hashTable\_setup3 | HashTableTest | The hash table has added values and some are collided |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Objective: test if the hastable is capable of detecting when adding is not possible due to a repeated key | | | | |
| **Class** | **Method** | **Scenery** | **Input value** | **Return** |
| HashTableTest | Add() | hashTable\_setUp2 | 0.4, "Alejadro Magno"  0.4, "MonitorGod" | "Exception object with the same key that other" |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Objective: test if the hastable is capable of adding an element if the list is empty | | | | |
| **Class** | **Method** | **Scenery** | **Input value** | **Return** |
| HashTableTest | add | hashTable\_setUp1 | 11.0, "Hola" | the element is added to the hash table |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Objective: test if the hash table is capable of detecting when searching is not possible due to being empty | | | | |
| **Class** | **Method** | **Scenery** | **Input value** | **Return** |
| HashTableTest | search | hashTable\_setUp1 | search(0,3) | Exception list is void |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Objective: test if the hash table is adding and removing effectively | | | | |
| **Class** | **Method** | **Scenery** | **Input value** | **Return** |
| HashTableTest | remove | hashTable\_setUp1 | 0.3, 0.4, 2, 2.4, 4.5, 6.5, 8.5. | hash table should be empty |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Objective: test if search method is working properly | | | | |
| **Class** | **Method** | **Scenery** | **Input value** | **Return** |
| HashTableTest | search | hashTable\_setUp2 | search(8.5) | se encuentra el valor buscado con la llave 0.4, en este caso “Juan” |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Objective: test if search method is working properly when there is a collision | | | | |
| **Class** | **Method** | **Scenery** | **Input value** | **Return** |
| HashTableTest | add  search | hashTable\_setUp2 | n = 0.39, "Marco Aurelio"  a = 0.37, "Alejandro Magno";  search(0.37) | 0.39, "Alejandro Magno" should be found |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Objective: test if search throws exception when the item doesn’t existe | | | | |
| **Class** | **Method** | **Scenery** | **Input value** | **Return** |
| HashTableTest | search | hashTable\_setUp2 | search(0.26) | Exception object doesn't exist |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Objective: test if remove method throws exception when the item doesn’t exist | | | | |
| **Class** | **Method** | **Scenery** | **Input value** | **Return** |
| HashTableTest | remove | hashTable\_setUp2 | remove(0.26) | Exception object doesn’t exist |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Objective: test if remove method works properly | | | | |
| **Class** | **Method** | **Scenery** | **Input value** | **Return** |
| HashTableTest | remove | hashTable\_setUp2 | remove(0.3) | the element should not exist in the hash table |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Objective: test if clone method makes identical items | | | | |
| **Class** | **Method** | **Scenery** | **Input value** | **Return** |
| HashTableTest | clone() | hashTable\_setUp2 |  | the elements should be the same in both hash tables |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Objective: Check if the elements are real clones and not pointer to the original items | | | | |
| **Class** | **Method** | **Scenery** | **Input value** | **Return** |
| HashTableTest | clone() | hashTable\_setUp2 |  | the elements should be the same in both hash tables and changing something in one of the elements doesn’t change the clone of the other hash table |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Objective: test the search method for an element that is collided | | | | |
| **Class** | **Method** | **Scenery** | **Input value** | **Return** |
| HashTableTest | search | hashTable\_setUp3 | 14.12 | the item should be found and equal to “Aurelio” |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Objective: test if remove works well when it is eliminating a collisioned element | | | | |
| **Class** | **Method** | **Scenery** | **Input value** | **Return** |
| HashTableTest | remove | hashTable\_setUp3 | 14.12 | the element with key 14.12 should be removed |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Objective: test if add method works well when it would make a collision | | | | |
| **Class** | **Method** | **Scenery** | **Input value** | **Return** |
| HashTableTest | add | hashTable\_setUp3 | 0.36, “Marco” | the element with key 14.12 should be removed |

# 

# Heap

|  |  |  |
| --- | --- | --- |
| **Name** | **Class** | **Scenery** |
| heap\_setup1 | HeapTest | An empty Heap. |
| heap\_setup2 | HeapTest | The Heap has some elements in it. |
| heap\_setup3 | HeapTest | The heap has some negative values in it |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Objective: correctly inserting an element in an empty Heap | | | | |
| **Class** | **Method** | **Scenery** | **Input value** | **Return** |
| Heap | Insert | heap\_setUp1 | a = 1, 10 | Using the method estactMax() we can confirm if the item is at the top of the list, meaning it was inserted. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Objective: Confirm the extractMax() method is working properly extracting some elements | | | | |
| **Class** | **Method** | **Scenery** | **Input value** | **Return** |
| Heap | extractMax() | heap\_setUp2 |  | Extracting three elements should return: 10, 20, 30 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Objective: Check if changing the priority of one elements would make it to be on top of the Heap | | | | |
| **Class** | **Method** | **Scenery** | **Input value** | **Return** |
| Heap | increase key()  extractMax() | heap\_setUp2 | IncreaseKey(10, 4) | changing the priority of one element would affect the extractMax()’s result. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Objective: Check if changing the priority of one elements would make it to be on top of the Heap | | | | |
| **Class** | **Method** | **Scenery** | **Input value** | **Return** |
| Heap | increase key()  extractMax() | heap\_setUp2 | IncreaseKey(10, 4) | changing the priority of one element would affect the extractMax()’s result, in this case it should be equal to 10. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Objective: Check if extractMax method returns null when the Heap is empty() | | | | |
| **Class** | **Method** | **Scenery** | **Input value** | **Return** |
| Heap | extractMax() | heap\_setUp1 |  | null |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Objective: Check if extractMax works even when the object is null | | | | |
| **Class** | **Method** | **Scenery** | **Input value** | **Return** |
| Heap | extractMax() | heap\_setUp1 |  | null |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Objective: Check if adding and extracting an element works | | | | |
| **Class** | **Method** | **Scenery** | **Input value** | **Return** |
| Heap | Insert  ExtractMax() | heap\_setUp1 | 1,10  10 | The same element must be found |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Objective: Check if extractMax extract in order | | | | |
| **Class** | **Method** | **Scenery** | **Input value** | **Return** |
| Heap | ExtractMax() | heap\_setUp2 |  | 40  30  20 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Objective: Inserting and extracting elements with high priority | | | | |
| **Class** | **Method** | **Scenery** | **Input value** | **Return** |
| Heap | insert | heap\_setUp1 | Integer.MaxValue, 10  Integer.Min, 20  Integer.MaxValue, 30 | when extracting elements, it should be in order |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Objective: Inserting and extracting | | | | |
| **Class** | **Method** | **Scenery** | **Input value** | **Return** |
| Heap | insert | heap\_setUp1 | 5, 10 | 10 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Objective: Inserting and extracting them in order | | | | |
| **Class** | **Method** | **Scenery** | **Input value** | **Return** |
| Heap | insert | heap\_setUp1 | 3, 10  2, 20  1, 30  4, 40 | Extracting them should return  40  10  20  30 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Objective: Inserting elements with the same priority and extracting them | | | | |
| **Class** | **Method** | **Scenery** | **Input value** | **Return** |
| Heap | insert | heap\_setUp1 | 1, 10  1, 20  1, 30  1, 40  1, 50 | Extracting them should give all of these elements.  10  50  40  30  20 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Objective: Test if cloning makes an identical Heap | | | | |
| **Class** | **Method** | **Scenery** | **Input value** | **Return** |
| Heap | insert | heap\_setUp1 | 1, 10  2, 20  3, 30 | Each of the elements should be the same in both Heaps. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Objective: test if changing the priority of a task throws an exception when the task doesn’t exist | | | | |
| **Class** | **Method** | **Scenery** | **Input value** | **Return** |
| Heap | Heap.increaseKey() | heap\_setUp2 | 15, 4 | throws exceptionThisDataStructureIsVoid. |

## Queue

|  |  |  |
| --- | --- | --- |
| **Name** | **Class** | **Scenery** |
| queue\_setup1 | QueueTest | Queue is empty |
| queue\_setup2 | QueueTest | The queue has some String values |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Objective: Test if adding in an empty queue works | | | | |
| **Class** | **Method** | **Scenery** | **Input value** | **Return** |
| QueueTest | offer | queue\_setUp1 | “Hola” | The element should be added to the queue |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Objective: Test if adding in an non empty queue works | | | | |
| **Class** | **Method** | **Scenery** | **Input value** | **Return** |
| QueueTest | offer | queue\_setUp2 | “Hola” | The element should be added to the queue |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Objective: Test if removing an element in an empty queue throws exception | | | | |
| **Class** | **Method** | **Scenery** | **Input value** | **Return** |
| QueueTest | poll | queue\_setUp1 |  | it should throw an exception of type : exceptionThisDataStructureIsVoid |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Objective: Test if removing an element in an non empty queue works | | | | |
| **Class** | **Method** | **Scenery** | **Input value** | **Return** |
| QueueTest | poll | queue\_setUp2 |  | it should return “Hola” |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Objective: Test if looking at the element in the front in an empty queue throws an exception | | | | |
| **Class** | **Method** | **Scenery** | **Input value** | **Return** |
| QueueTest | front | queue\_setUp1 |  | it should throw an exception of type : exceptionThisDataStructureIsVoid |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Objective: Test if looking at the element in the front in an non empty queue throws an exception | | | | |
| **Class** | **Method** | **Scenery** | **Input value** | **Return** |
| QueueTest | front | queue\_setUp2 |  | it should return “Hola” |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Objective: Test if eliminating every element in the queue makes it empty | | | | |
| **Class** | **Method** | **Scenery** | **Input value** | **Return** |
| QueueTest | empty() | queue\_setUp2 | 4\*poll() | queue.isEmpty() = true |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Objective: Test if the size() method return the quantity of elements that are in the queue | | | | |
| **Class** | **Method** | **Scenery** | **Input value** | **Return** |
| QueueTest | size() | queue\_setUp2 |  | queue.size() == 4 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Objective: Test if the clone() methods clones properly each of the elements of the queue | | | | |
| **Class** | **Method** | **Scenery** | **Input value** | **Return** |
| QueueTest | size() | queue\_setUp2 |  | queue.size() == newQueue.size()  for every element  queue.poll() == newQueue.poll()  or equals if it is an object |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Objective: Check if cloning doesn’t make a reference to the previous element, but a new object that is identical. Changing an element of the first queue and the same element in the cloned queue shouldn’t be identical. | | | | |
| **Class** | **Method** | **Scenery** | **Input value** | **Return** |
| QueueTest | size() | queue\_setUp2 |  | The items shouldn’t be identical |

## Stack

|  |  |  |
| --- | --- | --- |
| **Name** | **Class** | **Scenery** |
| stack\_setup1 | StackTest |  |
| stack\_setup2 | StackTest | The stack has some String values |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Objective: adding in an empty stack | | | | |
| **Class** | **Method** | **Scenery** | **Input value** | **Return** |
| StackTest | add() | stack\_setUp1 | “hola” | The items should be added |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Objective: adding in an non empty stack | | | | |
| **Class** | **Method** | **Scenery** | **Input value** | **Return** |
| StackTest | add() | stack\_setUp2 | “hola” | The items should be added |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Objective: removing an element in an empty stack | | | | |
| **Class** | **Method** | **Scenery** | **Input value** | **Return** |
| StackTest | pop() | stack\_setUp1 |  | throw an exceptionThisDataStructureIsVoid |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Objective: removing an element in an non empty stack | | | | |
| **Class** | **Method** | **Scenery** | **Input value** | **Return** |
| StackTest | pop() | stack\_setUp2 |  | must be equal to “amazing” |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Objective: looking at the top of an empty stack | | | | |
| **Class** | **Method** | **Scenery** | **Input value** | **Return** |
| StackTest | top() | stack\_setUp1 |  | must return null |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Objective: looking at the top of an non empty stack | | | | |
| **Class** | **Method** | **Scenery** | **Input value** | **Return** |
| StackTest | top() | stack\_setUp2 |  | must return “amazing” |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Objective: removing all elements must make the stack empty | | | | |
| **Class** | **Method** | **Scenery** | **Input value** | **Return** |
| StackTest | pop()  isEmpty() | stack\_setUp2 | pop() \* 5 | "Exception the list is void" |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Objective: test the size() method, it should return the same amount of elements added | | | | |
| **Class** | **Method** | **Scenery** | **Input value** | **Return** |
| StackTest | size() | stack\_setUp2 |  | must return 4 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Objective: test the size() method, it should return the same amount of elements added | | | | |
| **Class** | **Method** | **Scenery** | **Input value** | **Return** |
| StackTest | size() | stack\_setUp2 |  | must return 4 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Objective: test clone method. | | | | |
| **Class** | **Method** | **Scenery** | **Input value** | **Return** |
| StackTest | size() | stack\_setUp2 |  | the nes stack must have identical elements, but modifying one doesn’t affect the other. |

## Agenda

|  |  |  |
| --- | --- | --- |
| **Name** | **Class** | **Scenery** |
| Agenda\_setup1 | AgendaTest | Empty agenda |
| Agenda\_setup2 | AgendaTest | The agenda have some tasks |
| Agenda\_setup3 | AgendaTest | An arrayList with identical elements as the Agenda setup\_2 to compare |
| Agenda\_extremeSetup  Extreme case | AgendaTest | An agenda with lots of tasks.  Id, title, description, endline, priority  1, “Tittle ” 0, “Description “ 0, 00-00-0000, 0  2, “Tittle ” 1, “Description “ 1, 00-00-0000, 1  3, “Tittle ” 2, “Description “ 2, 00-00-0000, 2  4, “Tittle ” 3, “Description “ 3, 00-00-0000, 3  5, “Tittle ” 4, “Description “ 4, 00-00-0000, 0  …  1000, “Tittle ” 999, “Description “ 999, 00-00-0000, 0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Objective: adding in an empty Agenda | | | | |
| **Class** | **Method** | **Scenery** | **Input value** | **Return** |
| AgendaTest | add() | Agenda\_setUp1 | "Aristizabal Lord of the night", "I should finish my book", "22-02-2024", 2 | The item should be added |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Objective: adding in a non empty Agenda | | | | |
| **Class** | **Method** | **Scenery** | **Input value** | **Return** |
| AgendaTest | add() | Agenda\_setUp2 | "The questions for the monitors", "Jhoan and Richard are good guys", "22-02-2024", 4 | The item should be added |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Objective: searching in an empty agenda | | | | |
| **Class** | **Method** | **Scenery** | **Input value** | **Return** |
| AgendaTest | search() | Agenda\_setUp1 | 1 | exceptionThisDataStructureIsVoid |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Objective: searching a task that doesn’t exist | | | | |
| **Class** | **Method** | **Scenery** | **Input value** | **Return** |
| AgendaTest | search() | Agenda\_setUp2 | 100 | exceptionTheObjectDoesntExist |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Objective: searching a task that exist | | | | |
| **Class** | **Method** | **Scenery** | **Input value** | **Return** |
| AgendaTest | search() | Agenda\_setUp2 | 100 | exceptionTheObjectDoesntExist |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Objective: Remove a task in empty agenda | | | | |
| **Class** | **Method** | **Scenery** | **Input value** | **Return** |
| AgendaTest | removeTask() | Agenda\_setUp1 | 1  2 | “The data is empty in the priority task”  "The data is empty in the non-priority task" |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Objective: Remove a non priority task | | | | |
| **Class** | **Method** | **Scenery** | **Input value** | **Return** |
| AgendaTest | removeTask() | Agenda\_setUp2  Agenda\_setup3 | 2 | Removing a non priority task in agenda\_setup2 must be identical to the element in the second position of the arraylist used in agenda\_setup3 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Objective: Removing every priority task, every time the element removed must be the one with the biggest priority. | | | | |
| **Class** | **Method** | **Scenery** | **Input value** | **Return** |
| AgendaTest | removeTask() | Agenda\_setUp2  Agenda\_setup3 |  | Agenda\_setup3(7) == Agenda\_setup2.extractMax(1);  Agenda\_setup3(4) == Agenda\_setup2.extractMax(1);  Agenda\_setup3(0) == Agenda\_setup2.extractMax(1);  Agenda\_setup3(5) == Agenda\_setup2.extractMax(1);  Agenda\_setup3(1) == Agenda\_setup2.extractMax(1);  Agenda\_setup3(3) == Agenda\_setup2.extractMax(1); |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Objective: Removing every non priority task, every time the element removed must be the oldest one. | | | | |
| **Class** | **Method** | **Scenery** | **Input value** | **Return** |
| AgendaTest | removeTask() | Agenda\_setUp2  Agenda\_setup3 | 2  2  2 | Agenda\_setup3(6) == Agenda\_setup2.extractMax(2);  Agenda\_setup3(2) == Agenda\_setup2.extractMax(2);  "The data is empty in the non-priority task" |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Objective: Removing a task won’t eliminate others | | | | |
| **Class** | **Method** | **Scenery** | **Input value** | **Return** |
| AgendaTest | removeTask() | Agenda\_setUp2 | 7 | the rest of task should still exist |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Objective: modifying in empty Agenda | | | | |
| **Class** | **Method** | **Scenery** | **Input value** | **Return** |
| AgendaTest | modyfy() | Agenda\_setUp2 | 1,"Aristizabal Lord of the night", "I should finish my book", "22-02-2024", 2 | exceptionThisDataStructureIsVoid |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Objective: modifying in normal case | | | | |
| **Class** | **Method** | **Scenery** | **Input value** | **Return** |
| AgendaTest | modyfy() | Agenda\_setUp2 | 1, "Hi, I am German", " And the video at today", "22-02-2025", 3 | the task number 1, should be changed with this information, it can be confirmed by searching. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Objective: modifying and the item doesn’t exist | | | | |
| **Class** | **Method** | **Scenery** | **Input value** | **Return** |
| AgendaTest | modyfy() | Agenda\_setUp2 | 100, "Hi, I am German", " And the video at today", "22-02-2025", 3 | "The task was not modified because the task don’t exist" |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Objective: cloning and agenda makes identical items but not a reference to the same object | | | | |
| **Class** | **Method** | **Scenery** | **Input value** | **Return** |
| AgendaTest | modyfy() | Agenda\_setUp2 |  | “The agenda was cloned, different objects” |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Objective: changing a clone won’t affect the original agenda. | | | | |
| **Class** | **Method** | **Scenery** | **Input value** | **Return** |
| AgendaTest | modyfy() | Agenda\_setUp2 |  | the values must be different |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Objective: test if every of the elements are added | | | | |
| **Class** | **Method** | **Scenery** | **Input value** | **Return** |
| AgendaTest | search() | Agenda\_extremeSetUp | 0  1  2  …  999 | All te values must be found |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Objective: for every element cloned, the elements are equal but each has a different memory direction | | | | |
| **Class** | **Method** | **Scenery** | **Input value** | **Return** |
| AgendaTest | clone() | Agenda\_extremeSetUp |  | All te values must be identical but with a different memory direction |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Objective: changing every element should make different elements | | | | |
| **Class** | **Method** | **Scenery** | **Input value** | **Return** |
| AgendaTest | modify() | Agenda\_extremeSetUp | 0, “Hi, I am German “ 0, , " And the video at today "+i\*i, "22-02-2025", 3  1, “Hi, I am German “ 0, , " And the video at today "+i\*i, "22-02-2025", 3  …  999, “Hi, I am German “ 0, , " And the video at today "+i\*i, "22-02-2025", 3 | All te values must be modified |

## Task

|  |  |  |
| --- | --- | --- |
| **Name** | **Class** | **Scenery** |
| task\_setup1 | TaskTest | A task with this information:  12345678,"Marco Aurelio ","he owes me money","22-02-2034",2 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Objective: test if the constructor assigns the atributes properly | | | | |
| **Class** | **Method** | **Scenery** | **Input value** | **Return** |
| TaskTest | Task() | Task\_setup1 |  | All the atributes must be assigned |

## ControllerAgendaTest

|  |  |  |
| --- | --- | --- |
| **Name** | **Class** | **Scenery** |
| ControllerAgenda\_setup2 | ControllerAgenda | This tasks are added:  Id, title, description, endLine, priority  1, "Aristizabal Lord of the night", "I should finish my book", "22-02-2024", 2  2, "Meeting with Client", "Prepare presentation", "15-03-2024", 1  3, "Grocery Shopping", "Buy groceries for the week", "25-02-2024", 0  4, "Doctor's Appointment", "Checkup with Dr. Smith", "10-03-2024", 1  5, "Clean the House", "Vacuum and mop", "05-03-2024", 3 |
| ControllerAgenda\_setup1 | ControllerAgenda | An empty controller |
| ControllerAgenda\_setup3 | ControllerAgenda | Is the ControllerAgenda\_setup2 but with some modifications.  We have a new variable “agenda” in which we will save the differents versions of the agenda and an arrayList called “changes” in which we will save the text that is printed when doing an action. Then we do these changes:  control.removeTask(1);  changes.add("The task Submit Project Report was removed");  agenda.add(control.getAgenda().clone());  control.removeTask(2);  changes.add("The task Grocery Shopping was removed");  agenda.add(control.getAgenda().clone());  control.addTask("Plan Vacation", "Research and book flights", "30-04-2024", 4);  changes.add("Add new task: Plan Vacation");  agenda.add(control.getAgenda().clone());  control.addTask("Pay Bills", "Utility and credit card bills", "05-03-2024", 0);  changes.add("Add new task: Pay Bills");  agenda.add(control.getAgenda().clone());  control.modifyTask(3, "Call Mom", "Check-in with Mom", "01-03-2024", 0);  changes.add("modify task by title: Call Mom");  agenda.add(control.getAgenda().clone());  control.removeTask(1);  changes.add("The task Plan Vacation was removed");  agenda.add(control.getAgenda().clone());  control.removeTask(2);  changes.add("The task Workout was removed");  agenda.add(control.getAgenda().clone());  Ending like this: |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Objective: undoing an action when no action has been made | | | | |
| **Class** | **Method** | **Scenery** | **Input value** | **Return** |
| ControllerAgenda | undo () | ControllerAgenda\_setup1 |  | “There is no more versions to undo" |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Objective: undoing in an agenda should make a different agenda for every undo() call | | | | |
| **Class** | **Method** | **Scenery** | **Input value** | **Return** |
| ControllerAgenda | undo () | ControllerAgenda\_setup2 |  | Each time the agendas must be different |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Objective: every element after undoing an action must be the same | | | | |
| **Class** | **Method** | **Scenery** | **Input value** | **Return** |
| ControllerAgenda | undo () | ControllerAgenda\_setup2 |  | Each time the elements must be the same, unless the action affected the task |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Objective: undoing in an agenda should make a different agenda for every undo() call | | | | |
| **Class** | **Method** | **Scenery** | **Input value** | **Return** |
| ControllerAgenda | undo () | ControllerAgenda\_setup2 |  | "The Agenda versions should be different by its changes: " |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Objective: getting every element of the AgendaController should be equal to evey element added, it can be confirmed by using an array of the previous elements | | | | |
| **Class** | **Method** | **Scenery** | **Input value** | **Return** |
| ControllerAgenda | toString () | ControllerAgenda\_setup2 |  | Each element should be the same |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Objective: assert that undoing an action return the system to the exact previous version | | | | |
| **Class** | **Method** | **Scenery** | **Input value** | **Return** |
| ControllerAgenda | toString () | ControllerAgenda\_setup3 |  | Every version should be the same when undoing an action, and also the messages that are printed when doing an action |