**THE ENGINEERING DESIGN METHOD** \*

**PHASE 1: PROBLEM IDENTIFICATION**

The problem of optimizing the process of passenger check-in and check-out is a constant concern for airlines, and this project specifically seeks to improve the registration of passenger information and the order in which they must enter the aircraft. To achieve this, an efficient system must be created that allows large amounts of data to be loaded and prioritizes first class information in the boarding process. This system must also be able to reward passenger punctuality and improve the efficiency and punctuality of the airline.

To meet the needs of the project, it is essential to develop a system that allows loading passenger information corresponding to a flight and efficiently searching for complete information of passengers arriving at the boarding lounge. This system must be able to register the arrival of passengers and rewards their punctuality by admitting them to the aircraft in the order of arrival, following the call by sections of the aircraft.

In addition, it is necessary to design a special system for first class that takes into account other relevant data, such as accumulated miles, special attention required, third age, among others, and that allows these passengers to be prioritized in the boarding process.

To meet the project requirements efficiently and effectively, the following five general requirements have been identified:

1. Develop a system for reading data recorded in a file: this system will allow loading passenger information corresponding to a flight and efficiently search for complete information on passengers arriving at the boarding lounge. This will ensure punctuality and efficiency in the boarding process.
2. Allow passengers to check-in to the check-in queue: the system must allow passengers to check-in to a waiting queue, sorted by arrival time. This allows an effective and orderly organization of passengers before boarding.
3. Establish an effective organization of passengers on a first-come, first-served basis: an effective organization of passengers who have registered their arrival must be ensured, so that they are assigned a fair and efficient order of entry to the aircraft.
4. Allow passengers to enter passengers by sections and priority: to improve the efficiency of the boarding process, it is necessary to allow passengers to enter passengers by sections and priorities, based on special rules, such as ticket type, miles accrued, special attention required, seniority or other relevant data.

**PHASE 2: GATHERING THE NECESSARY INFORMATION**

Once the specific needs and requirements of the problem have been established, the engineer begins to collect relevant information and data needed to solve the problem. In this process, it is important to consider the type of information needed and the appropriate techniques for its collection, which will depend on the nature of the problem at hand.

For example, in designing an efficient and effective system to improve the passenger experience during the boarding process, it is essential to take into account the needs of passengers traveling with small children during the boarding and departure process. It is important to take into account the different policies and regulations established by airlines for these passengers, which may affect the boarding order. Therefore, it is essential that the system developed for the boarding process considers these specific needs and regulations to ensure a smoother and more comfortable travel experience for all passengers.

In the investigation of the problem, the search for bibliographic information is a fundamental support in this phase. The literature review allows knowing what others have learned about related problems, which can be very useful for the engineer in the selection of the appropriate techniques and tools to solve the problem in question.

Iberia's boarding page provides detailed information on the boarding process for the airline's flights, taking into account the needs of customers requiring special assistance. In conclusion, it is essential to implement a solution that allows the efficient and organized management of passenger check-in and check-out, dividing them by sections according to the priority that corresponds in each case. To do this, it is necessary to consider the specific needs of each moment and each type of section. Therefore, it is necessary to find a quick, simple and effective solution that ensures a pleasant and comfortable experience for passengers.

**PHASE 3: SEARCH FOR CREATIVE SOLUTIONS**

By way of introduction, we identified 4 subproblems of the system, which are: defining the aircraft distribution, establishing a departure order, an arrival order and establishing a passenger data storage system. Based on the above, we generated a matrix that relates the identified component or subproblem associated with its possible solutions or ideas, as shown in the following table:

[Brainstorming table by component.](https://docs.google.com/spreadsheets/d/172V98pIPvZ2VUzhQYceQe8SyoLuRZ5TkOGuxyY4M_CE/edit?usp=sharing)

**PHASE 4: TRANSITION FROM BRAINSTORMING TO PRELIMINARY DESIGNS**

The first thing we do in this step is to discard the ideas that are not feasible per component, to finally arrive at a conglomerate of best solutions by subproblems. In this sense we have the following discards and their reasons.

| Description of the alternative | Reason for discarding |
| --- | --- |
| **The first class section will be located at the end of the aircraft.** | What is sought after in upper class is comfort, which makes it counterproductive for first class users to have to travel a greater distance in relation to standard class to enter and exit the aircraft. |
| **Input order: Filling by columns** | Filling by columns implies establishing a fixed order of entry, without taking into account priority factors such as punctuality, since, if a fixed order is not established, entry through a single aisle can generate traffic and disorder when different passengers enter in different rows but in the same column. |
| **Data storage: Boost insertion speed.** | Because the system revolves around the flight passengers, the insertion and dropout flow is minimal, therefore, it is not relevant to enhance dynamic data insertion. |

Now, taking into account the remaining solution components, we generate two alternatives, which allow us to design and visualize the possible solutions of the system. We start from the following alternatives:

**Alternative 1:**

The storage of passenger data will be done in a hashTable, which enhances the immediate query of passengers due to its addressing with its hash function. The hashtable performs the passenger search through an ID, which allows the system to load the passenger arrival order by manipulating only their unique id and not a whole object with the passenger information. The information can be accessed through this ID at a later time.

The entrance will be made starting from the sections or group of rows furthest away from the entrance until reaching the closest area. It should be noted that different sections will be created according to passenger class, i.e. standard and first class. In this case, the sections are taken by groups of rows to reward the punctuality of the passengers to a greater extent, since having a greater number of seats, there will necessarily be a greater chance that a passenger will enter faster depending on his punctuality.

The ordering of incoming passengers would be done through the heapSort algorithm, which implies that a priority queue will be used. This queue will be organized according to a priority coefficient generated from the punctuality and other attributes of each passenger. This allows factors such as accumulated miles, special attention required, seniority or other relevant data to be taken into account when generating the order of entry, enriching the application.

The order of departure would be from the sections closest to the exit, where the passenger close to the aisle will depart first. Between each pair of passengers close to the aisle, the one who has been more punctual will exit first. It should be noted that this order is given by a priority queue by means of a heapSort.

This alternative implies that punctuality and other factors that determine priority will be relevant in the order of filling each section. That is, punctuality and other factors are respected within each section or cluster of rows and you will not have problems such as passengers who arrive early, but are in a row close to the gate, would enter later than those who arrive late, but are seated in rows farther away from the gate that fill up first.

**Alternative 2**

The storage of the passenger data will be done in an arrayList that allows the search of the passenger through an ID, which allows the system to have a loading of the passenger arrival order by manipulating only its unique id and not a whole object with the passenger information. The information can be accessed through this ID at a later time.

The entrance, in this case, will be through the rows farthest from the entrance until reaching the nearest zone. In this case, punctuality is rewarded only with respect to one row, thus reducing the importance of the punctuality factor in the system.

The ordering of incoming passengers would be done through the heapSort algorithm, which implies that a priority queue will be used. This queue will be organized according to a priority coefficient generated from the punctuality and other attributes of each passenger. This allows factors such as accumulated miles, special attention required, seniority or other relevant data to be taken into account when generating the order of entry, enriching the application.

The order of departure would be from the rows closest to the exit, where the passenger who leaves first will be the one who has been the most punctual, regardless of his proximity to the aisle. This allows rewarding punctuality at the cost of greater traffic when leaving the aircraft, since at the time of departure a passenger may have to overtake other passengers who have not been evacuated. It should be noted that this order is given by a priority queue through a heapSort.

This alternative implies that punctuality and other factors that determine priority will be relevant in the order in which each row is filled. But the order of queue entry will remain the same, which means that passengers who arrive early, but are in a queue close to the gate, would enter later than those who arrive late, but are seated in rows farther away from the gate that fill up first.

**PHASE 5: EVALUATION AND SELECTION OF THE BEST SOLUTION**

**NOTE: THIS EVALUATION WILL BE DONE ON MODULES WITH AN ASSIGNED PERCENTAGE, MODULES = (Aircraft Distribution, Order of Entry, Order of Exit, Data Storage)**

**Criterion A**, the proposed solution delivers a solution that meets the requirements:

[5] Completely, [3] Partially, [1] With difficulties

**Criterion B**, the proposed solution is optimal in a real context:

[5] Completely, [3] Partially, [1] With difficulties

**Criterion C**, the proposed solution has a complexity in the worst case.

[5] Constant [4] Logarithmic [3] Linear [2] N Logarithmic [1] Quadratic [2] N Logarithmic [1] Quadratic

[0] Exponential

Click [here](https://docs.google.com/spreadsheets/d/172V98pIPvZ2VUzhQYceQe8SyoLuRZ5TkOGuxyY4M_CE/edit?usp=sharing) to view the rating in detail and look at the comments in the cells. See page 2.

| **IDEA / COMPONENT** | **Aircraft distribution** | **Order of entry** | **Order of departure** | **Data storage and access** |  |
| --- | --- | --- | --- | --- | --- |
| **IDEA 1 - Final** | 10% | 20% | 30% | 40% |  |
| **The proposed solution delivers a solution that meets the following requirements** | 5 | 5 | 5 | 5 |  |
| **The proposed solution is optimal in a real-world context.** | 5 | 5 | 5 | 5 |  |
| **Time complexity** | Not applicable | 2 | 2 | 5 | **TOTAL** |
| **Score** | 5 | 4 | 4 | 5 | 4.5 |

| **IDEA / COMPONENT** | **Aircraft distribution** | **Order of entry** | **Order of departure** | **Data storage and access** |  |
| --- | --- | --- | --- | --- | --- |
| **IDEA 2** | 10% | 20% | 30% | 40% |  |
| **The proposed solution delivers a solution that meets the following requirements** | 5 | 5 | 5 | 5 |  |
| **The proposed solution is optimal in a real-world context.** | 3 | 3 | 3 | 5 |  |
| **Time complexity** | Not applicable | 2 | 2 | 2 | **TOTAL** |
| **Score** | 4 | 3.333333333 | 3.333333333 | 4 | 3.666666667 |

**PHASE 6: PREPARATION OF REPORTS AND SPECIFICATIONS**

Problem: Handling the entry and exit of an airplane.

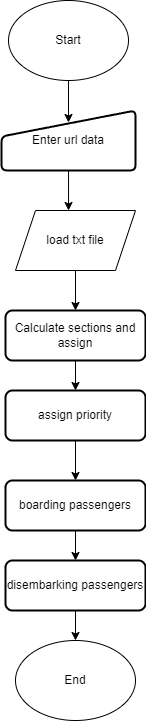
Inputs: txt file of passenger input in Json format

Departures: Order of departure and order of arrival

**Pseudocode:**

****

**Flowchart:**

****

**PHASE 7: DESIGN IMPLEMENTATION**

**Implementation in a Programming Language (Java)**

**List of tasks to be implemented:**

**a. Load passenger list and order of entry**

**b. Calculate passenger cross-section**

**c. Set priority coefficient**

**d. Generate passenger hashTable**

**e. Generate input sort order**

**f. Generate output sort order**

**Subroutines**

| **METHOD:** | **outPutOrdering** |
| --- | --- |
| **ENTRANCE:** | **None** |
| **OUTPUT:** | **String out, represents the departure order of the passengers.** |
| **DESCRIPTION:** | **This method generates a report with the departure order of passengers, according to their priority in the waiting list. To do this, first a list of hash table nodes representing the passengers is obtained, then a priority heap is created and the passenger nodes are inserted into it, sorting them according to their type (first class or standard class) and their priority. The heap is then sorted using the heapSort() method and the output report is generated by traversing the sorted heap.** |

| **public** String **outPutOrdering**) {   ArrayList<HashNode<String,Passenger>> passengers = elementList();   Heap<Priority, String> outputHeap =**new**Heap<>();  **for** (**int**i = 0; i < passengers.size(); i++) {   HeapNode<Priority, String> node =**null** **if** (passengers.get(i).getValue() **instanceof**FirstClassPassenger){   outputHeap.insert((((FirstClassPassenger) passengers.get(i).getValue()).getPriority(),  (passengers.get(i).getValue()).getPassengerID()));   }**else if** (passengers.get(i).getValue() **instanceof**StandardPassenger) {   outputHeap.insert((((StandardPassenger) passengers.get(i).getValue()).getPriority(),  (passengers.get(i).getValue()).getPassengerID());  }  }  //Ordering   outputHeap.heapSort();  // Generating report   String chain = ""  **for** (**int** i = outputHeap.getList().size() - 1, j = 0; i >= 0; i--) chain += (outputHeap.getList().size() - i) + ". " + outputHeap.getList().get(i).getValue() + "\n"  **return**chain; } |
| --- |

| **METHOD:** | **entrySort()** |
| --- | --- |
| **ENTRANCE:** | **None** |
| **OUTPUT:** | **String out, represents the order of entry of the passengers.** |
| **DESCRIPTION:** | **This method uses a Heap to sort the Hash nodes by passenger entry priority, which is calculated by the calculatePriority() method of the Passenger class. After inserting the nodes into the Heap, the heapSort() method is used to sort them and a string is generated with the sorted list.** |

| **public String entrySort**){   ArrayList<HashNode<String,Passenger>> passengers = elementList();  Heap<Double, String> entryOrderPassenger =**newHeap<Double, String>();   // This "for" is used to insert the heap nodes with K = totalPriority and V = id. int totalStandardSections = (int**Math.ceil((getRows()-getRowsFirstClass()) / Math.ceil((getRows()-getRowsFirstClass())/10));  **for (inti = 0; i < passengers.size(); i++) entryOrderPassenger.insert(passengers.get(i).getValue().calculatePriority(totalStandardSections),passengers.get(i).getKey());  //Ordering   entryOrderPassenger.heapSort();  // Generating report   String chain = ""**  **for (inti = entryOrderPassenger.getList().size() - 1; i >= 0; i--) {  chain += (entryOrderPassenger.getList().size() - i) +". " + entryOrderPassenger.getList().get(i).getValue() + "\n"**  } **returnchain; }** |
| --- |

| **METHOD:** | **load(String path)** |
| --- | --- |
| **ENTRANCE:** | **String path, document link** |
| **OUTPUT:** | **None** |
| **DESCRIPTION:** | **This method loads passenger data from a JSON formatted file in the path specified by "path". First, it reads the contents of the file and stores it in a string. Then it uses the Gson library to deserialize the JSON objects into instances of the Passenger class. Next, it sets the priorities of each passenger and assigns them a section based on their type (first class or standard). Finally, it creates a hash table to store the Passenger objects.** |

| **public void load**(String path) **throws**IOException {     File file =**new**File(path);   FileInputStream fis =**new**FileInputStream(file);  BufferedReader reader =**new** BufferedReader(**new**InputStreamReader(fis));  String content =""  String line ="" **while**((line = reader.readLine()) != **null**{  content += line +"\n"  }  // Gson with types   Gson gson gson = **new**GsonBuilder()  .registerTypeAdapter(Passenger.class,**new**Serializer<Passenger>()).create();    Passenger passengers [] = gson.fromJson(content, Passenger[].class);   ArrayList<Passenger> passengersList =**new**ArrayList<>();  passengersList.addAll(List.of(passengers));  //Priorities Staff   establishPriorities(passengersList);  sectionCalculationAndAssignment(passengersList);  //Creating hash table   generatePassengersHasTable(passengersList);  } |
| --- |

| **METHOD:** | **sectionCalculationAndAssignment(ArrayList<Passenger> passengers)** |
| --- | --- |
| **ENTRANCE:** | **ArrayList<Passenger> passengers** |
| **OUTPUT:** | **None** |
| **DESCRIPTION:** | **This method calculates and assigns a section value to each passenger, depending on their passenger type (first class or standard) and the rows of seats on the aircraft.** |

| **public void sectionCalculationAndAssignment**(ArrayList<Passenger> passengers){  // Section Calculation. **double** sectionFirstClass = Math.ceil( (**double**getRowsFirstClass() / 10); **double** sectionStandardClass = Math.ceil((**double**(getRows()-getRowsFirstClass())/10);  // This "for" is used to assign a section value for each passenger depending on its type, first class or Standard. **for** (**int**i = 0; i < passengers.size(); i++) { **if** (passengers.get(i) **instanceof**FirstClassPassenger){   passengers.get(i).setSection((**int**sectionFirstClass);   }**else if** (passengers.get(i) **instanceof**StandardPassenger) {  passengers.get(i).setSection(**int**sectionStandardClass);  }  } } } |
| --- |

| **METHOD:** | **establishPriorities(ArrayList<Passenger> passengers)** |
| --- | --- |
| **ENTRANCE:** | **ArrayList<Passenger> passengers** |
| **OUTPUT:** | **None** |
| **DESCRIPTION:** | **This method goes through the ArrayList of passengers and establishes the priorities of each passenger based on their distance to the center of the aircraft column and their punctuality at check-in. To do this, it uses the "establishDistanceToCenter" and "establishPunctuality" methods of the Passenger class. These priorities will be used later in the program to sort passengers in the boarding queue.** |

| **public void establishPriorities**(ArrayList<Passenger> passengers){ **int**center = getColumns()/2; **int**numPassengers = passengers.size();  **for** (**int**i = 0; i < passengers.size(); i++) { **char**column = passengers.get(i).getTicket().charAt(0);   passengers.get(i).establishDistanceToCenter(center,column);  passengers.get(i).establishPunctuality(numPassengers,i+1 );   } } |
| --- |

| **public**ArrayList<HashNode<String,Passenger>> elementList(){  ArrayList<HashNode<String,Passenger>> elements =**new**ArrayList<>(); **for** (**int**i = 0; i < getPassengerHashTable().getListOfNodes().length; i++) { **if** (getPassengerHashTable().getListOfNodes()[i] != **null**{  collisionsElements(elements, getPassengerHashTable().getListOfNodes()[i]);  }  } **return**elements; } |
| --- |

| **METHOD:** | **generatePassengersHasTable(ArrayList<Passenger> passengers)** |
| --- | --- |
| **ENTRANCE:** | **ArrayList<Passenger> passengers** |
| **OUTPUT:** | **None** |
| **DESCRIPTION:** | **This method creates a hash table to store Passenger objects based on their Passenger ID. For each passenger in the input ArrayList, the insert() method of the hash table is called, which inserts a key-value pair into the table. The key is the passenger ID and the value is the corresponding Passenger object.** |

| **public void generatePassengersHasTable**(ArrayList<Passenger> passengers){ **this**.passengerHashTable = **new**HashTable<>(passengers.size()); **for** (**int**i = 0; i < passengers.size(); i++) { **this**.passengerHashTable.insert(passengers.get(i).getPassengerID(),passengers.get(i));  } } } |
| --- |

Link to gitHub:

<https://github.com/Sebastian-411/integrativeTaskCED.git>

**Complexity analysis of the entrySort() algorithm**

The entrySort() algorithm is composed of 4 main methods:

1. **elementList()** : Which organizes all the passengers of the heapSort into an arrayList.
2. **heapNodesInsertion()**: HeapNodes insertion of passenger IDs and priority into the heap.
3. **heapSort()**: Sorting by means of heapSort.
4. **report()**: The creation of the input order report.

| **Method ID** | **Time complexity** |
| --- | --- |
| elementList() |  |
| heapNodesInsertion() |  |
| heapSort() |  |
| report() |  |
| **TOTAL** |  |

**Complexity analysis of outPutOrdering() algorithm**

The outPutOrdering() algorithm is composed of 4 main methods:

1. **elementList()** : Which organizes all the passengers of the heapSort into an arrayList.
2. **heapNodesInsertion()**: HeapNodes insertion of passenger IDs and priority into the heap.
3. **heapSort()**: Sorting by means of heapSort.
4. **report()**: The creation of the input order report.

| **Method ID** | **Time complexity** |
| --- | --- |
| elementList() |  |
| heapNodesInsertion() |  |
| heapSort() |  |
| report() |  |
| **TOTAL** |  |

**Codes**

**Main methods:**

| **public** String **outPutOrdering**) {   ArrayList<HashNode<String,Passenger>> passengers = elementList();   Heap<Priority, String> outputHeap =**new**Heap<>();  **for** (**int**i = 0; i < passengers.size(); i++) {   HeapNode<Priority, String> node =**null** **if** (passengers.get(i).getValue() **instanceof**FirstClassPassenger){   outputHeap.insert((((FirstClassPassenger) passengers.get(i).getValue()).getPriority(),  (passengers.get(i).getValue()).getPassengerID()));   }**else if** (passengers.get(i).getValue() **instanceof**StandardPassenger) {   outputHeap.insert((((StandardPassenger) passengers.get(i).getValue()).getPriority(),  (passengers.get(i).getValue()).getPassengerID());  }  }  //Ordering   outputHeap.heapSort();  // Generating report   String chain = ""  **for** (**int** i = outputHeap.getList().size() - 1, j = 0; i >= 0; i--) chain += (outputHeap.getList().size() - i) + ". " + outputHeap.getList().get(i).getValue() + "\n"  **return**chain; } |
| --- |

| **public String entrySort**){   ArrayList<HashNode<String,Passenger>> passengers = elementList();  Heap<Double, String> entryOrderPassenger =**newHeap<Double, String>();   // This "for" is used to insert the heap nodes with K = totalPriority and V = id. int totalStandardSections = (int**Math.ceil((getRows()-getRowsFirstClass()) / Math.ceil((getRows()-getRowsFirstClass())/10));  **for (inti = 0; i < passengers.size(); i++) entryOrderPassenger.insert(passengers.get(i).getValue().calculatePriority(totalStandardSections),passengers.get(i).getKey());  //Ordering   entryOrderPassenger.heapSort();  // Generating report   String chain = ""**  **for (inti = entryOrderPassenger.getList().size() - 1; i >= 0; i--) {  chain += (entryOrderPassenger.getList().size() - i) +". " + entryOrderPassenger.getList().get(i).getValue() + "\n"**  } **returnchain; }** |
| --- |

**Auxiliary Methods:**

| **publicArrayList<HashNode<String,Passenger>> elementList(){  ArrayList<HashNode<String,Passenger>> elements =newArrayList<>(); for (inti = 0; i < getPassengerHashTable().getListOfNodes().length; i++) { if (getPassengerHashTable().getListOfNodes()[i] != null**{  collisionsElements(elements, getPassengerHashTable().getListOfNodes()[i]);  }  } **returnelements; }**  **private void collisionsElements**(ArrayList<HashNode<String, Passenger>> elementList, HashNode<String,Passenger> current){ **if** (current == **null**) **return**  elementList.add(current);  collisionsElements(elementList, current.getNext()); } |
| --- |

**Monticule / Priority Queue Methods :**

| **public void heapSort**){  buildHeap(); **for** (**int**i = list.size()-1; i >= 1 ; i--) {  HeapNode temporary = list.get(0);  list.set(0,list.get(i));  list.set(i, temporary);  heapSize=1;  maxHeapify(0);  }  } |
| --- |

| **public void maxHeapify**(**int**from){ **int**left = getLeft(from); **int**right = getRigth(from); **int**largest = from;  **if** (left < h  eapSize){ **if**( list.get(left).getKey().compareTo(list.get(from).getKey().getKey()) > 0 )largest = left;  }  **if**(right < heapSize){ **if**( list.get(right).getKey().compareTo(list.get(largest).getKey()) > 0 ) largest = right;  }  **if**if (largest != from){  HeapNode temporal = list.get(from);  list.set(from, list.get(largest));  list.set(largest, temporal);  maxHeapify(largest); } } }  **public void buildHeap**){ **this**.heapSize = list.size(); **for** (**int**i = (list.size() /2) -1; i >=0 ; i--) {  maxHeapify(i);  } } |
| --- |

**TAD tables**

[**Click here to view the TAD tables**](https://docs.google.com/spreadsheets/d/1mCf1uQCuUTffIWb-rNL9hJ1HDPU0dqw7lmKReOcaSVU/edit?usp=sharing)

**Testing:**

**HashTable**

| **Name** | **Class** | **Scenario** |
| --- | --- | --- |
| **setupStage1** | **HashTableTest** | **An object of class HashTable with ARR\_SIZE = 5.** |
| **setupStage2** | **HashTableTest** | **An object of class HashTable with ARR\_SIZE = 5.**  **It contains 12 HashNodes with attributes:**  **- key = "AAAA", value = 1**  **- key = "BBCC", value = 2**  **- key = "CCDD", value = 3**  **- key = "DDEE", value = 4**  **- key = "EEFF", value = 5**  **- key = "FFGG", value = 6**  **- key = "GGHH", value = 7**  **- key = "HHII", value = 8**  **- key = "IIJJ", value = 9**  **- key = "JJKK", value = 10**  **- key = "KKLL", value = 11**  **- key = "LLMM", value = 12** |

| **Test Objective: Verify that the insert method of the HashTable class works correctly, adding the element created from the key and value parameters.** | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenario** | **Input Values** | **Result** |
| **HashTable** | **insert** | **setupStage1** | **key = "AAAA"**  **value = 1** | **A new node has been inserted in the HashTable chained to the node at position 0 of the array.** |
| **HashTable** | **insert** | **setupStage1** | **key = "BBCC" key = "BBCC" key = "BBCC" key = "BBCC" key = "BBCC**  **value = 2** | **A new node has been inserted in the HashTable at position 1 of the array.** |
| **HashTable** | **insert** | **setupStage1** | **key = "CCDD" key = "CCDD"**  **value = 3** | **A new node has been inserted in the HashTable at position 0 of the array.** |

| **Test Objective: Verify that the insert and search methods of the HashTable class work correctly, adding the element created from the key and value parameters and then search the element from the key provided.** | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenario** | **Input Values** | **Result** |
| **HashTable** | **insert** | **setupStage1** | **key = "AAAA"**  **value = 1** | **A new node has been inserted in the HashTable chained to the node at position 0 of the array.** |
| **HashTable** | **insert** | **setupStage1** | **key = "BBCC" key = "BBCC" key = "BBCC" key = "BBCC" key = "BBCC**  **value = 2** | **A new node has been inserted in the HashTable at position 1 of the array.** |
| **HashTable** | **insert** | **setupStage1** | **key = "CCDD" key = "CCDD"**  **value = 3** | **A new node has been inserted in the HashTable at position 0 of the array.** |
| **HashTable** | **search** | **setupStage1** | **key = "CCDD" key = "CCDD"** | **The value related to the key passed as parameter is obtained. Returns the value 3.** |
| **HashTable** | **search** | **setupStage1** | **key = " BBCC "** | **The value related to the key passed as parameter is obtained. Returns the value 2.** |
| **HashTable** | **search** | **setupStage1** | **key = " AAAA "** | **The value related to the key passed as parameter is obtained. Returns the value 1.** |

| **Test Objective: Verify that the insert, search and delete methods of the HashTable class work correctly, adding the element created from the key and value parameters, searching the element from the key, deleting the element using its key and searching it through the same key returning nil.** | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenario** | **Input Values** | **Result** |
| **HashTable** | **insert** | **setupStage1** | **key = "AAAA"**  **value = 1** | **A new node has been inserted in the HashTable chained to the node at position 0 of the array.** |
| **HashTable** | **search** | **setupStage1** | **key = " AAAA "** | **The value related to the key passed as parameter is obtained. Returns the value 1.** |
| **HashTable** | **delete** | **setupStage1** | **key = " AAAA "** | **A boolean value True is obtained.** |
| **HashTable** | **search** | **setupStage1** | **key = " AAAA "** | **Null is obtained since the node was eliminated with the key provided.** |

| **Test Objective: Verify that the insert method of the HashTable class works correctly, adding the element created from the key and value parameters.** | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenario** | **Input Values** | **Result** |
| **HashTable** | **insert** | **setupStage1** | **key = "AAAA"**  **value = 1** | **A new node has been inserted in the HashTable chained to the linkedList at position 0 of the array.** |
| **HashTable** | **insert** | **setupStage1** | **key = "BBCC" key = "BBCC" key = "BBCC" key = "BBCC" key = "BBCC**  **value = 2** | **A new node has been inserted in the HashTable chained to the linkedList at position 1 of the array.** |
| **HashTable** | **insert** | **setupStage1** | **key = "CCDD" key = "CCDD" key = "CCDD" key = "CCDD**  **value = 3** | **A new node has been inserted in the HashTable chained to the linkedList at position 0 of the array.** |
| **HashTable** | **insert** | **setupStage1** | **key = "DDEE"**  **value = 4** | **A new node has been inserted in the HashTable chained to the linkedList at position 4 of the array.** |
| **HashTable** | **insert** | **setupStage1** | **key = "EEFF" key = "EEFF" key = "EEFF" key = "EEFF" key = "EEFF**  **value = 5** | **A new node has been inserted in the HashTable chained to the linkedList at position 3 of the array.** |
| **HashTable** | **insert** | **setupStage1** | **key = "FFGG"**  **value = 6** | **A new node has been inserted in the HashTable chained to the linkedList at position 2 of the array.** |
| **HashTable** | **insert** | **setupStage1** | **key = "GGHH"**  **value = 7** | **A new node has been inserted in the HashTable chained to the linkedList at position 1 of the array.** |
| **HashTable** | **insert** | **setupStage1** | **key = "HHII"**  **value = 8** | **A new node has been inserted in the HashTable at position 0 of the array.** |
| **HashTable** | **insert** | **setupStage1** | **key = "IIJJ"**  **value = 9** | **A new node has been inserted in the HashTable at position 4 of the array.** |
| **HashTable** | **insert** | **setupStage1** | **key = "JJKK"**  **value = 10** | **A new node has been inserted in the HashTable at position 3 of the array.** |
| **HashTable** | **insert** | **setupStage1** | **key = "KKLL" key = "KKLL"**  **value = 11** | **A new node has been inserted in the HashTable at position 2 of the array.** |
| **HashTable** | **insert** | **setupStage1** | **key = "LLMM"**  **value = 12** | **A new node has been inserted in the HashTable at position 1 of the array.** |

| **Test Objective: Verify that the search methods of the HashTable class work correctly, searching for the element from the key parameters and return the value if found or nil otherwise.** | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenario** | **Input Values** | **Result** |
| **HashTable** | **search** | **setupStage2** | **key = "AAAA"** | **The value related to the key passed as parameter is obtained. Returns the value 1.** |
| **HashTable** | **search** | **setupStage2** | **key = "BBCC" key = "BBCC" key = "BBCC" key = "BBCC" key = "BBCC** | **The value related to the key passed as parameter is obtained. Returns the value 2.** |
| **HashTable** | **search** | **setupStage2** | **key = "CCDD" key = "CCDD"** | **The value related to the key passed as parameter is obtained. Returns the value 3.** |
| **HashTable** | **search** | **setupStage2** | **key = "DDEE"** | **The value related to the key passed as parameter is obtained. Returns the value 4.** |
| **HashTable** | **search** | **setupStage2** | **key = "EEFF" key = "EEFF" key = "EEFF" key = "EEFF" key = "EEFF** | **The value related to the key passed as parameter is obtained. Returns the value 5.** |
| **HashTable** | **search** | **setupStage2** | **key = "FFGG"** | **The value related to the key passed as parameter is obtained. Returns the value 6.** |
| **HashTable** | **search** | **setupStage2** | **key = "GGHH"** | **The value related to the key passed as parameter is obtained. Returns the value 7.** |
| **HashTable** | **search** | **setupStage2** | **key = "HHII"** | **The value related to the key passed as parameter is obtained. Returns the value 8.** |
| **HashTable** | **search** | **setupStage2** | **key = "IIJJ"** | **The value related to the key passed as parameter is obtained. Returns the value 9.** |
| **HashTable** | **search** | **setupStage2** | **key = "JJKK"** | **The value related to the key passed as parameter is obtained. Returns the value 10.** |
| **HashTable** | **search** | **setupStage2** | **key = "KKLL" key = "KKLL"** | **The value related to the key passed as parameter is obtained. Returns the value 11.** |
| **HashTable** | **search** | **setupStage2** | **key = "LLMM"** | **The value related to the key passed as parameter is obtained. Returns the value 12.** |

| **Test Objective: Verify that the delete and search methods of the HashTable class work correctly, deleting the element created from the key provided. Subsequently return nil when searching for the deleted element.** | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenario** | **Input Values** | **Result** |
| **HashTable** | **delete** | **setupStage2** | **key = "AAAA"** | **A boolean value True is obtained.** |
| **HashTable** | **search** | **setupStage2** | **key = "AAAA"** | **Null is obtained since the node was eliminated with the key provided.** |
| **HashTable** | **delete** | **setupStage2** | **key = "BBCC" key = "BBCC" key = "BBCC" key = "BBCC" key = "BBCC** | **A boolean value True is obtained.** |
| **HashTable** | **search** | **setupStage2** | **key = "BBCC" key = "BBCC" key = "BBCC" key = "BBCC" key = "BBCC** | **Null is obtained since the node was eliminated with the key provided.** |
| **HashTable** | **delete** | **setupStage2** | **key = "CCDD" key = "CCDD"** | **A boolean value True is obtained.** |
| **HashTable** | **search** | **setupStage2** | **key = "CCDD" key = "CCDD"** | **Null is obtained since the node was eliminated with the key provided.** |
| **HashTable** | **delete** | **setupStage2** | **key = "DDEE"** | **A boolean value True is obtained.** |
| **HashTable** | **search** | **setupStage2** | **key = "DDEE"** | **Null is obtained since the node was eliminated with the key provided.** |
| **HashTable** | **delete** | **setupStage2** | **key = "EEFF" key = "EEFF" key = "EEFF" key = "EEFF" key = "EEFF** | **A boolean value True is obtained.** |
| **HashTable** | **search** | **setupStage2** | **key = "EEFF" key = "EEFF" key = "EEFF" key = "EEFF" key = "EEFF** | **Null is obtained since the node was eliminated with the key provided.** |
| **HashTable** | **delete** | **setupStage2** | **key = "FFGG"** | **A boolean value True is obtained.** |
| **HashTable** | **search** | **setupStage2** | **key = "FFGG"** | **Null is obtained since the node was eliminated with the key provided.** |
| **HashTable** | **delete** | **setupStage2** | **key = "GGHH"** | **A boolean value True is obtained.** |
| **HashTable** | **search** | **setupStage2** | **key = "GGHH"** | **Null is obtained since the node was eliminated with the key provided.** |
| **HashTable** | **delete** | **setupStage2** | **key = "HHII"** | **A boolean value True is obtained.** |
| **HashTable** | **search** | **setupStage2** | **key = "HHII"** | **Null is obtained since the node was eliminated with the key provided.** |
| **HashTable** | **delete** | **setupStage2** | **key = "IIJJ"** | **A boolean value True is obtained.** |
| **HashTable** | **search** | **setupStage2** | **key = "IIJJ"** | **Null is obtained since the node was eliminated with the key provided.** |
| **HashTable** | **delete** | **setupStage2** | **key = "JJKK"** | **A boolean value True is obtained.** |
| **HashTable** | **search** | **setupStage2** | **key = "JJKK"** | **Null is obtained since the node was eliminated with the key provided.** |
| **HashTable** | **delete** | **setupStage2** | **key = "KKLL" key = "KKLL"** | **A boolean value True is obtained.** |
| **HashTable** | **search** | **setupStage2** | **key = "KKLL" key = "KKLL"** | **Null is obtained since the node was eliminated with the key provided.** |
| **HashTable** | **delete** | **setupStage2** | **key = "LLMM"** | **A boolean value True is obtained.** |
| **HashTable** | **search** | **setupStage2** | **key = "LLMM"** | **Null is obtained since the node was eliminated with the key provided.** |

| **Test Objective: Verify that the insert, search and delete methods of the HashTable class work correctly in table boundary conditions. The objective is to add elements with key and value provided, find the elements with a parameterized key and then delete those elements found. Following this, it is expected that nil is returned when searching for deleted elements.** | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenario** | **Input Values** | **Result** |
| **HashTable** | **insert** | **setupStage2** | **i will take values from 0 to 99.**  **key = "" + i+i+i+i+i**  **value = i\*5** | **Insertion of 100 nodes to the hashTable.** |
| **HashTable** | **search** | **setupStage2** | **key = "IIJJ"** | **The value related to the key passed as parameter is obtained. Returns the value 9.** |
| **HashTable** | **search** | **setupStage2** | **key = "JJKK"** | **The value related to the key passed as parameter is obtained. Returns the value 10.** |
| **HashTable** | **search** | **setupStage2** | **key = "KKLL" key = "KKLL"** | **The value related to the key passed as parameter is obtained. Returns the value 11.** |
| **HashTable** | **search** | **setupStage2** | **key = "LLMM"** | **The value related to the key passed as parameter is obtained. Returns the value 12.** |
| **HashTable** | **search** | **setupStage2** | **key = "6666666666"** | **The value related to the key passed as parameter is obtained. Returns the value 330.** |
| **HashTable** | **search** | **setupStage2** | **key = "1111"** | **The value related to the key passed as parameter is obtained. Returns the value 5.** |
| **HashTable** | **delete** | **setupStage2** | **key = "6666666666"** | **A boolean value True is obtained.** |
| **HashTable** | **search** | **setupStage2** | **key = "6666666666"** | **Null is obtained since the node was eliminated with the key provided.** |
| **HashTable** | **search** | **setupStage2** | **key = "14141414"** | **The value related to the key passed as parameter is obtained. Returns the value 70.** |
| **HashTable** | **delete** | **setupStage2** | **key = "GGHH"** | **A boolean value True is obtained.** |
| **HashTable** | **search** | **setupStage2** | **key = "GGHH"** | **Null is obtained since the node was eliminated with the key provided.** |
| **HashTable** | **delete** | **setupStage2** | **key = "14141414"** | **A boolean value True is obtained.** |
| **HashTable** | **search** | **setupStage2** | **key = "14141414"** | **Null is obtained since the node was eliminated with the key provided.** |
| **HashTable** | **search** | **setupStage2** | **key = "BBCC" key = "BBCC" key = "BBCC" key = "BBCC" key = "BBCC** | **The value related to the key passed as parameter is obtained. Returns the value 2.** |
| **HashTable** | **delete** | **setupStage2** | **key = " BBCC "** | **A boolean value True is obtained.** |
| **HashTable** | **search** | **setupStage2** | **key = " BBCC "** | **Null is obtained since the node was eliminated with the key provided.** |
| **HashTable** | **insert** | **setupStage2** | **i will take values from 0 to 99.**  **key = "" + i+i+i+i+i+i+i**  **value = i\*10** | **Insertion of 100 nodes to the hashTable.** |
| **HashTable** | **delete** | **setupStage2** | **key = " 14141414141414"** | **A boolean value True is obtained.** |
| **HashTable** | **search** | **setupStage2** | **key = " 14141414141414"** | **Null is obtained since the node was eliminated with the key provided.** |
| **HashTable** | **search** | **setupStage2** | **key = "CCDD" key = "CCDD"** | **The value related to the key passed as parameter is obtained. Returns the value 3.** |
| **HashTable** | **search** | **setupStage2** | **key = "DDEE"** | **The value related to the key passed as parameter is obtained. Returns the value 4.** |
| **HashTable** | **search** | **setupStage2** | **key = "EEFF" key = "EEFF" key = "EEFF" key = "EEFF" key = "EEFF** | **The value related to the key passed as parameter is obtained. Returns the value 5.** |
| **HashTable** | **search** | **setupStage2** | **key = " FFGG"** | **The value related to the key passed as parameter is obtained. Returns the value 6.** |
| **HashTable** | **search** | **setupStage2** | **key = " 6666666666666666"** | **The value related to the key passed as parameter is obtained. Returns the value 660.** |
| **HashTable** | **search** | **setupStage2** | **key = " 55555"** | **The value related to the key passed as parameter is obtained. Returns the value 50.** |
| **HashTable** | **delete** | **setupStage2** | **key = "6666666666666666"** | **A boolean value True is obtained.** |
| **HashTable** | **search** | **setupStage2** | **key = "6666666666666666"** | **Null is obtained since the node was eliminated with the key provided.** |
| **HashTable** | **search** | **setupStage2** | **key = " 151515151515"** | **The value related to the key passed as parameter is obtained. Returns the value 150.** |
| **HashTable** | **delete** | **setupStage2** | **key = "LLMM"** | **A boolean value True is obtained.** |
| **HashTable** | **search** | **setupStage2** | **key = "LLMM"** | **Null is obtained since the node was eliminated with the key provided.** |
| **HashTable** | **delete** | **setupStage2** | **key = "14141414"** | **A Boolean value False is obtained.** |
| **HashTable** | **search** | **setupStage2** | **key = "14141414"** | **Null is obtained since the node was eliminated with the key provided.** |
| **HashTable** | **search** | **setupStage2** | **key = " 93939393"** | **The value related to the key passed as parameter is obtained. Returns the value 465.** |
| **HashTable** | **delete** | **setupStage2** | **key = "93939393"** | **A boolean value True is obtained.** |
| **HashTable** | **search** | **setupStage2** | **key = "93939393"** | **Null is obtained since the node was eliminated with the key provided.** |

**Heap:**

| **Name** | **Class** | **Scenario** |
| --- | --- | --- |
| setUpStage1 | HeapTest | Initialize An empty heap object. |
| setUpStage2 | **HeapTest** | This code sets up a Heap data structure with three HeapNode elements containing integer and string values.  Nodes: <12, "1">, <5, "1">, <6, "1"> |
| setUpStage3 | HeapTest | This code sets up a Heap data structure with 6 HeapNode elements containing integer and string values.    HeapNode<>(4, "1"), HeapNode<>(14, "2")HeapNode<>(7, "3"), HeapNode<>(2, "4")), HeapNode<>(8, "5"), HeapNode<>(1, "6") |
| setUpStage4 | HeapTest | This code initializes a Heap data structure with ten HeapNode elements, where each node contains an integer and a string value. Here are the nodes that have been added:  Integer value 4 and string value "1".  Integer value 1 and string value "2".  Integer value 3 and string value "3".  Integer value 2 and string value "4".  Integer value 16 and string value "5".  Integer value 9 and string value "6".  Integer value 10 and string value "7".  Integer value 14 and string value "8".  Integer value 8 and string value "9".  Integer value 7 and string value "10". |
| setUpStage5() | HeapTest | This code initializes a Heap data structure with ten HeapNode elements, where each node contains an integer and a string value. Here are the nodes that have been added:  Integer value 5 and string value "1".  Integer value 9 and string value "2"  Integer value 10 and string value "3"  Integer value15 and string value "4" |

| **Purpose :** Ensure the correctness of insert, buildHeap, Heapify and heapsort functioning, | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenario** | **Input Values** | **Expected result** |
| Heap | insert() | setUpStage1() | K = 23, V = "1" | The node insert must be the new root |
| Heap | insert() | setUpStage2() | K = 15, V = "newRoot". | The node insert must be the new root.  The heapsize must increase in 1.  The node k = 12 must be at 1 position. |
| Heap | insert() | setUpStage2 | K = -10, V= "leaf". | The heapsize must increase in 1.  The node k = 5 must be the parent of the new node.  The new node must be at 3 position. |
| Heap | maxHeapify() | setUpStage3() | from = 0 | The max node must be 14.  The node at 3 position must be 8.  The node at 4 position must be 4. |
| Heap | maxHeapify | setUpStage4() | from = 2 | The node at 6 position must be 3.  The node at 2 position must be 10. |
| Heap | maxHeapify | setUpStage4() | from = 3 | The node at 3 position must be 14.  The node at 7 position must be 2. |
| Heap | buildheap | setUpStage5() |  | The list must be ordered in this way = 15, 9, 10, 5. |
| Heap | buildheap | setUpStage5() |  | The list must be ordered in this way = 16, 14, 10, 8, 7, 9, 3, 2, 4, 1. |
| Heap | buildHeap | setUpStage2() |  | The list must be ordered in this way = 12, 5, 6 |
| Heap | heapSort() | setUpstage4 |  | The list must be ordered in this way = "1 2 3 4 7 8 9 10 14 16". |
| Heap | heapSort() | setUpStage5 |  | The list must be ordered in this way = "5 9 19 15". |

**Sources:**

[**Traveling with children on the plane - Iberia España**](https://www.iberia.com/es/viajar-con-iberia/ninos-y-bebes/)

[**Boarding procedure - Iberia Spain**](https://www.iberia.com/es/embarque/)

[**Priority boarding | Air Europa Spain**](https://www.aireuropa.com/es/es/aea/aexperience/servicios/priority-boarding.html)