**Method engineering**

**Step 1: Problem identification**

A well-known airline wants to improve its service to customers. That is why you want a program that, based on a database, allows you to register the passengers of a flight and organize them according to their priority. This, guaranteeing efficiency and speed in the search for records. Users are organized by sections, those who are in the first rows (first class), will be prioritized in the boarding process on the plane. In addition, the airline also wants to optimize the departure process, so that it is more comfortable and faster for all users who use the service.

**Requirements:**

| Client | The airline |
| --- | --- |
| User | The airline staff. |
| Functional requirements | The system must allow to:  R1: Load DataBase.  R2: Register Passenger.  R3:Show entry order.  R4: Show out order. |
| Context | A renowned airline company aims to enhance its customer service. To achieve this, they are seeking a program that can register flight passengers and organize them based on their priority using a database that ensures efficiency and quick retrieval of records. Users will be organized into sections, with those seated in the front rows (first-class) being given priority during the boarding process. Additionally, the airline intends to streamline the departure process to make it more comfortable and quicker for all passengers using their services. |
| Process Requirements | * R1: The program must be developed into teams of a maximum of 3 people. * R2. The system must be uploaded to the version control system GitHub, and the name of the repository should be in lowercase English with hyphens separating words if necessary. This repository must be kept private until the final launch * R3. Each progress or commit that is uploaded to the GitHub repository must have a difference of one hour, completing a minimum of 10 commits. |
| Non-functional requirements | * R1: The application interface must be simple and easy to understand so that users feel comfortable with it. * R2. The program must be developed in Java. |

| Name or identifier | R1. Load DataBase. | | |
| --- | --- | --- | --- |
| Summary | The system must allow loading a database with pre-generated information regarding the flight to be organized. | | |
| Inputs | Input name | DataType | Selection or repetition condition |
| dataFlight | txt |  |
| General activities necessary to get the results | The system will receive a database with the information of the airplane responsible for the flight and the data of passengers associated with this.  After that, the system will deserialize the database using Gson and will create different objects “Passenger”, allowing the program to make different actions with them.  Finally, the program will add these objects “Passenger” in a hash table in order to ease the process of searching and registering. | | |
| Result or postcondition | The database has been loaded successfully. | | |
| Outputs | Output name | DataType | Selection or repetition condition |
| message | String |  |

| Name or identifier | R2. Register Passenger. | | |
| --- | --- | --- | --- |
| Summary | Using another database, the system must register the information of the people who arrive to the flight, (the people who are in the waiting room) it also has to ensure maximum efficiency in searching and registering passengers for a given flight. | | |
| Inputs | Input name | DataType | Condición de selección o repetición |
| infoPassengers | txt |  |
| General activities necessary to get the results | The system will deserialize a database that has the id of the users that are waiting for the plane.  Using the hash table, the system must search the passenger associated with that id and it will add in the priority taking into account his features. | | |
| Result or postcondition | The passengers has been registered | | |
| Outputs  (There´s no outputs, the system just register the passengers) | Output name | DataType | Selection or repetition condition |
|  |  |  |

| Name or identifier | R3. Show entry order | | |
| --- | --- | --- | --- |
| Summary | The system must allow the responsible airline member to display the order in which passengers should enter. The order of entry will be according to each user's priority and; in case of a draw, the user who will enter first will be the first who has arrived at the waiting room. | | |
| Inputs  (There’s no inputs, the program does an internal process) | Input name | DataType | Selection or repetition condition |
|  |  |  |
| General activities necessary to get the results | Once all the passengers are registered and ordered in the priority queue, using the method extracr, the program will show to the user the order in which the passengers are entering the airplane. | | |
| Result or postcondition | The entry order has been shown by screen. | | |
| Outputs | Input name | DataType | Selection or repetition condition |
| enterOrder | String |  |

| Name or identifier | R4. Show out order | | |
| --- | --- | --- | --- |
| Summary | The system must allow the responsible airline member to display the order in which passengers should exit the plane. The order of exit will prioritize the people who are in the first rows (first class) and in every row the person who gets out first will be the nearest to the aisle. In case of a draw, will get out first the person who have reached their seats first. | | |
| Inputs  (There’s no inputs, the program does an internal process) | Input name | DataType | Selection or repetition condition |
|  |  |  |
| General activities necessary to get the results | Once the flight arrives at its destination, the passengers will be put in a priority queue according to the organization of the airplane and the criteria explained previously. After that, in order to guarantee that the person who left the plane first, all the passengers will be put on a stack. Now, using the method of dequeue, the crew could see the way in which passengers overflow the plane. | | |
| Result or postcondition | The exit order has been shown. | | |
| Outputs | Input name | DataType | Selection or repetition condition |
| exitOrder | String |  |

**Step 2: Information gathering:**

**Related problem:**

**Case Lufthansa:**

According to (Ariza, 2019). The Lufthansa airline has always sought to optimize time in order to prepare the plane for takeoff as soon as possible. Once the passenger is at the boarding gate, with the help of announcements, each passenger in the group is told the place where they should board. However, many times, being of another nationality, or simply because they want to board, they do not follow the instructions and enter when they see that the boarding has already begun. This causes delays in the check-in line, disorder in the boarding area and inconvenience to passengers.

**Ariza, proposed the following solution to the problem:**



\*For him, it is pertinent to organize the boarding process by groups as shown in the image.

Source:

<https://alejandria.poligran.edu.co/bitstream/handle/10823/1639/Plan%20de%20mejoramiento%20de%20la%20organizaci%c3%b3n%20en%20la%20sala%20de%20abordaje%20del%20vuelo%20operado%20por%20Lufthansa%20en%20la%20estaci%c3%b3n%20de%20Bogot%c3%a1..pdf?sequence=1&isAllowed=y>

**Meanings:**

Key concepts for understanding the problem and finding solutions.

**Priorization:**

Prioritization is the process of adding a value to a task, feature, or scenario and then ranking from most to least important based on that assigned value.

<https://apliint.com/2022/03/01/que-es-la-priorizacion-y-como-se-utiliza-en-el-desarrollo-de-aplicaciones/>

**Organization:**

The way in which a system is arranged to achieve the desired results is known as organization.

<https://www.significados.com/organizacion/#:~:text=Se%20conoce%20como%20organizaci%C3%B3n%20a,para%20lograr%20alg%C3%BAn%20prop%C3%B3sito%20espec%C3%ADfico>.

**Customer satisfaction:**

Satisfaction is a measure of how well the products supplied and the services provided by a company meet or exceed customer expectations.

<https://www.itaerea.es/atencion-cliente-aeropuertos>

**Boarding:**

Operation of boarding an aircraft in order to start a flight.

<https://dpej.rae.es/lema/embarque#:~:text=Adm.,anteriores%20del%20mismo%20vuelo%20directo>.

**Travel categories: (On a plane)**

There are two categories: first class and economy class:

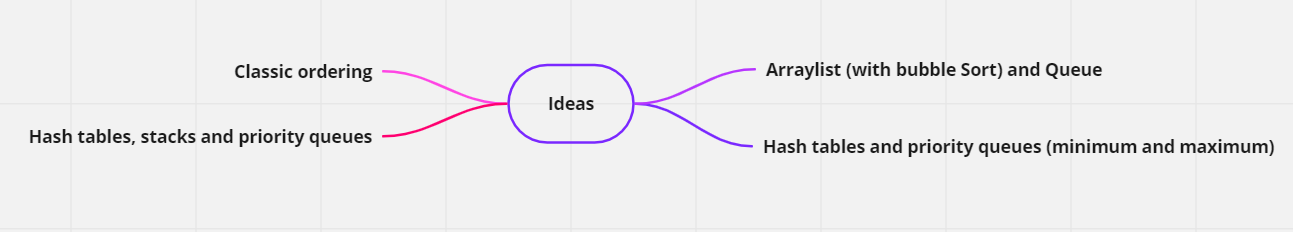
In economy class, passengers traveling in economy class are entitled to basic services.

In the first class, the first class has more facilities, comfort and benefits.

<https://topadventure.com/viajes/Primera-clase-y-clase-turista-en-avion-que-diferencias-tienen-20220706-0009.html>

**Step 3: Search for creative solutions:**

**Brainstorm:**



**Alternative 1:**

**Classic ordering:**

One of the solutions that could be used to solve the problem is the use of paper and pencil.

The use of these tools makes it possible to record the information of the people who arrive at the boarding gate and later, use this collection to organize the departure process. With a list, the entry of each of the passengers could be recorded. From this list, the aircraft boarding order could be built and this order could be taken into account to guarantee the correct deboarding process. Although it can be a bit difficult, with a correct organization, all the requirements of the problem situation could be satisfied.

**Alternative 2:**

**Hash tables, stacks and priority queues:**

Another alternative solution could be the use of hash tables, stacks and priority queues.

In the problem, it is specified that the record lookup must be done in the fastest and most optimal way possible, which is why the implementation of the hash table would be one of the best options. On the other hand, prioritization is key to solving this problem, which is why the priority queue could be a good option. Certain characteristics must be taken into account, such as: In the event that it is first class: If you are an older adult, if you have accumulated miles, if you require special attention or (as a tie-breaking factor) the order of arrival. On the other hand, it could also be used as an intermediary for the overflow process. If the users are ordered prioritizing the opposite, that is, prioritizing the passenger who is far from the aisle and who was the last to sit down. This would allow the implementation of the stack, which through its pop method, will guarantee the order in the output process.

**meanings:**

***Hash Table:*** The hash table is an associative data structure that relates a key and a value using the function called hash, this function allows us to calculate the index where the elements will go.

***Stack:****The stack is a linear data structure which allows both adding and removing elements from the head (LIFO).*

***Priority Queue:*** The queue is a linear data structure which allows adding elements at the end and removing elements from the head (FIFO), however, in this case, as it is a priority, for its ordering, the value of priority of the element to enter. That's why its implementation is done using a heap.

**Alternative 3:**

**Hash tables and priority queues (minimum and maximum):**

Another solution to the problem could be the implementation of a hash table together with priority queues, the same entry process will be handled, users are entered in the hash table to later organize themselves on the plane taking into account their class, priority (if that it is first class) and later its order of arrival. The difference is that for the departure process, the passengers will be ordered in a queue of minimum priority, prioritizing first class passengers (since they are those who are in the first rows), ordering first in the queue those closest to the aisle, using order of arrival as a tie-breaker.

**Alternative 4:**

**Arraylist (with bubble Sort) and Queue:**

In the problem, the data structure to use is not specified, so one might think that the ArrayList could be a good option to store information. Methods of this class could easily allow operations with the incoming database. The order for the approaching process can be solved by implementing ordering methods on the structure, such as bubbleSort. Which would ensure the priority of passengers. On the other hand, the queue would be key to the onboarding process, since once organized on the plane, the queue could fill up with the passengers from the first rows to the passengers from the last rows (with a tie-break factor for whoever was first). has arrived). Finally, with the dequeue method, the correct order in the output process would be guaranteed.

**Meanings:**

***ArrayList:*** The ArrayList is a data structure that allows data to be stored in memory in a similar way to Arrays, the difference between these two is that for the arraylist, the number of elements it stores could be called unlimited, that is, it does not require declaring its size.

***Queue:*** The queue is a linear data structure which follows the FIFO principle, that is, it allows adding elements at the end and removing elements from the head.

**Step 4: Transitioning ideas to preliminary designs**

**Classic ordering:**

The pen and paper alternative would satisfy all the requirements, but would be tedious and exhausting for the crew. On the other hand, this solution would be unmaintainable and harmful to the environment, since a certain amount of leaves would have to be used for each flight, which cannot be recovered later. For all this, it could be said that the alternative "Pencil and paper" would not be very convenient to solve the problem.

**Hash tables, stacks and priority queues:**

The alternative of hash table, stacks and priority queues could solve the problem without major difficulties. First of all, priority queues would make it possible to guarantee the requirement that the most “important” people are the first to go through the boarding process. On the other hand, its implementation together with the stack would guarantee the order in the output process from unstacking and following the divide-and-conquer principle, involving multiple data structures. For all this, it could be said that a good option at the time of implementation could be the use of stacks and priority queues.

**Hash tables and priority queues (minimum and maximum):**

The hash table and priority queues alternative might also solve the problem. It is similar to the previous alternative and, like the latter, it would make it possible to guarantee compliance with the requirements easily and quickly. In addition, we can say that this solution has certain advantages, such as avoiding the use of the intermediate priority queue and a stack, this could be an advantage in the implementation process.

**Arraylist (with bubble Sort) and Queue:**

The alternative ArrayList and Queues could also be to solve the problem with multiple facilities, its implementation guarantees the priority order in the boarding and unboarding process. Its only difficulty would be that it is much more expensive in terms of memory, if a traditional sorting method is implemented, such as bubbleSort, the complexity of the implementation could be raised to the order of O(n^2).

**Step 5: Evaluation and selection of the best solution.**

**The criteria to be evaluated for the selection of the solution:**

**Criterion 1:**

**Fulfillment of requirements:** The proposed solution must comply in the most optimal way with all the requirements.

**[2] Meets the requirements in the most optimal way.**

**[1] It meets the requirements but not in the most optimal way.**

**Criterion 2:**

**Divide and conquer:** A good solution is one that manages to divide a large problem into smaller and easier to handle problems, dividing tasks to achieve it. For this reason, the greater the number of data structures, the better.

**[2] Requires 3 or more data structures**

**[1] Requires less than 3 data structures**

**Criterion 3:**

**Complexity:** The proposed solution must allow it to be the one that takes the least amount of time to solve the user's requests, either in the search or the user registration process, since in the future the airline expects to work with a large amount of information.

**[4] Less than linear complexity**

**[3] Linear complexity**

**[2] Complexity greater than linear**

**[1] Its temporal complexity cannot be measured in terms of algorithms.**

|  | **Criterion 1** | **Criterion 2** | **Criterion 3** | **Total** |
| --- | --- | --- | --- | --- |
| **Alternative 1:**  **Classic ordering** | **1** | **2** | **1** | **4** |
| **Alternative 2:**  **Hash tables, stacks and priority queues** | **2** | **2** | **3** | **7** |
| **Alternative 3:**  **Hash tables and priority queues (minimum and maximum)** | **2** | **1** | **3** | **6** |
| **Alternative 4:**  **Arraylist (with bubble Sort) and Queue** | **1** | **2** | **2** | **5** |

For all of the above, the solution that obtained the best score was alternative (2), therefore, it is the one that can best be implemented as a solution.

**Step 6: Preparation of reports and specifications:**

**Problem Specification (in terms of input/output):**

**Problem:** Improve the efficiency and speed of searching records and organize passengers by providing a better service.

**Input:** Database in a plain text file.

**Output:** Location order to enter and exit the plane according to priorities.

**Considerations:**

The following cases must be taken into account to establish the order of passenger entry and exit location:

1. If the person is over 50 years old and belongs to the first class, they are given priority in the queue when entering.
2. For first class, the miles traveled with the airline are taken into account
3. The order of arrival will work as a “tiebreaker” factor in case two or more first class people have the same priority and as the only priority factor for all second class passengers.
4. The plane seats will be even and equal on each side of the aisle, this in order to seek to balance the weight of the plane as much as possible.
5. Each person will have their own seat regardless of the class they belong to, that is, there will not be two or more people in one seat.

**Structures’ TADS:**

**Stack**

| **TAD Stack** |
| --- |
| **Abstract Object: Stack** |
| **{Inv:**  **\*** 𝟎 ≤ 𝒏^𝑺𝒊𝒛𝒆  \* **en = Top(Stack)**  **\* ∀ e1, e2,.., en ∈ Stack, en > en-1 > … > e2 > e1 → en is the first to leave.**  **}** |
| **Primitive Operations:**  **Modifiers:**   * **push: O object X Stack -> Stack** * **pop: Stack -> Stack**   **Analyzers:**   * **top: Stack -> first element** |

**Modifiers:**

| **Push** |
| --- |
| **Adds a new element e to stack s** |
| **{𝒑𝒓𝒆: 𝒔 = < 𝒆𝟏, 𝒆𝟐, 𝒆𝟑, … , 𝒆𝒏 > ∧ 𝒆 𝒐𝒓 𝒔 = ∅}** |
| **{𝒑𝒐𝒔𝒕: 𝒔 = < 𝒆𝟏, 𝒆𝟐, 𝒆𝟑, … , 𝒆𝒏, e>}** |

| **Pop** |
| --- |
| **Extracts from the stack s; the last element** |
| **{𝒑𝒓𝒆: 𝒔 ≠ ∅ ∧ s = <e1,..., en-1, en> }** |
| **{𝒑𝒐𝒔𝒕: 𝒔 = < 𝒆𝟏, 𝒆𝟐, 𝒆𝟑, … , 𝒆𝒏−𝟏 >}** |

**Analyzer:**

| **Top** |
| --- |
| **Show the last element from Stack s.** |
| **{𝒑𝒓𝒆: 𝒔 ≠ ∅ ∧ s = <e1,..., en-1, en> }** |
| **{𝒑𝒐𝒔𝒕: 𝒔(Top) = en}** |

**PriorityQueue:**

| **TAD Priority Queue:** |
| --- |
| **Abstract Object: priorityQueue** |
| **{Inv: ∀ p,q, …, r ∈ P, priority(p) > priority(q) > … > priority(r) → p is the first to leave.**  **}** |
| **Primitive Operations:**  **Modifiers:**   * **enqueue: O object X priorityQueue -> priorityQueue** * **dequeue: priorityQueue -> first element**   **Analyzers:**   * **top: priorityQueue -> first element** * **last: priorityQueue -> last element** |

**Modifiers:**

| **Enqueue** |
| --- |
| **Insert a new element e to the back of the queue q** |
| **{𝒑𝒓𝒆: 𝒒 = < 𝒆𝟏, 𝒆𝟐, 𝒆𝟑, … , 𝒆𝒏 > ∧ 𝒆 𝒐𝒓 𝒒 = ∅}** |
| **{𝒑𝒐𝒔𝒕: 𝒒 =< 𝒆𝟏, 𝒆𝟐, 𝒆𝟑, … , 𝒆𝒏, 𝒆 >** |

| **Dequeue** |
| --- |
| **Extracts the element in Queue q’s front** |
| **{𝒑𝒓𝒆: 𝒒 ≠ ∅, ∧ 𝒒 = < 𝒆𝟏, 𝒆𝟐, 𝒆𝟑, … , 𝒆𝒏 >}** |
| **{𝒑𝒐𝒔𝒕: 𝒒 = < 𝒆𝟏, 𝒆𝟐, 𝒆𝟑, … , 𝒆𝒏−𝟏 > }** |

**Analyzer:**

| **Top** |
| --- |
| **Show the first element from queue q.** |
| **{𝒑𝒓𝒆: q ≠ ∅ ∧ q= <e1,..., en-1, en> }** |
| **{𝒑𝒐𝒔𝒕: q(Top) = e1}** |

| **Last** |
| --- |
| **Show the last element from queue q.** |
| **{𝒑𝒓𝒆: q ≠ ∅ ∧ q= <e1,..., en-1, en> }** |
| **{𝒑𝒐𝒔𝒕: q(Last) = en}** |

**Hash Table:**

| **TAD Hash Table** |
| --- |
| **Abstract Object: HashTable** |
| **{Inv: (∀𝒌 ∈ 𝑼niverse ∧ 𝒌 ∈ Keys) → 𝒉(𝒌) ∈ T)}** |
| **Primitive Operations:**  **Modifiers:**   * **insert: K key X V value X HashTable -> index** * **delete: HasTable x key -> position in HashTable**   **Analyzers:**   * **search: K key X HashTable -> element** |

**Modifiers:**

| **Insert** |
| --- |
| **Inserts a new item v into a HashTable h with its Key k.** |
| **{𝒑𝒓𝒆: 𝒉 ≠ 𝜽 ∧ 𝑲𝒆𝒚 𝒌 ≠ 𝜽}** |
| **{pos: 𝒉 =< 𝒌, 𝒗 > }** |

| **Delete** |
| --- |
| **Deletes an item v with a given search key k from the HashTable h.** |
| **{𝒑𝒓𝒆: 𝒉 = ≪< 𝒌𝟎, 𝒗𝟎 >, … , < 𝒌𝒏, 𝒗𝒏 >≫ ∧ 𝑲𝒆𝒚 𝒌 ≠ 𝜽}** |
| **{pos: 𝒉 = ≪< 𝒌𝟎, 𝒗𝟎 >, … , < 𝒌𝒏-1, 𝒗𝒏-1 >≫ }** |

**Analyzer:**

| **Search** |
| --- |
| **Retrieves an item v with a given search key k from a HashTable h.** |
| **{𝒑𝒓𝒆: 𝒉 ≠ 𝜽 𝒂𝒏𝒅 𝑲𝒆𝒚 𝒌 ≠ 𝜽}** |
| **{𝒑𝒐𝒔𝒕: 𝒉 ≠ 𝜽 𝒂𝒏𝒅 𝒓𝒆𝒕𝒖𝒓𝒏 < 𝒌, 𝒗 > }** |

**Heap:**

| **TAD Heap** |
| --- |
| **Abstract Object: Heap** |
| **{Inv:**  **\*max Heap: max(Tree) = root(Tree)**  **\*min Heap: min(Tree) = root(Tree)**  **\*∀ b1, b2 ∈ Branches, length(b1) = h ∧ length(b2) = h-1 → b1 and b2 are not adjacent}** |
| **Primitive Operations:**  **Modifiers:**   * **insert: Object X Heap -> Heap** * **delete: Heap X object -> Heap**   **Analyzers:**   * **getRoot: Heap -> Heap(root)** * **size: Heap -> Heap(Size)** |

**Modifiers:**

| **Insert** |
| --- |
| **Inserts a new item v into a heap h.** |
| **{𝒑𝒓𝒆: 𝒉 ≠ 𝜽}** |
| **{pos: 𝒉 =< 𝒗 > }** |

| **Delete** |
| --- |
| **Deletes an item v from the heap h.** |
| **{𝒑𝒓𝒆: 𝒉 = <𝒗𝟎, … , 𝒗𝒏 >}** |
| **{pos: 𝒉 = <𝒗𝟎, … , 𝒗𝒏-1 > ∧ h(size) = size-1 }** |

**Analyzer:**

| **GetRoot** |
| --- |
| **Return the first element e from the heap** |
| **{𝒑𝒓𝒆: 𝒉 ≠ 𝜽}** |
| **{pos: h(root)}** |

| **Size** |
| --- |
| **Return the amount of elements of the heap h** |
| **{𝒑𝒓𝒆: 𝒉 = <𝒗𝟎, … , 𝒗𝒏 >}** |
| **{pos: h(size) = n}** |

**Test cases:**

**\*For all the stages, exist a global object Plane, its features are the next:**

**(name=”Aereo”,rows=15,seatsByRows=4,firstClassRows=6”)**

| **Name** | **Class** | **Stage** |
| --- | --- | --- |
| SetupStage1 | AirLineTest | An object of type AirPlane without passengers or normal passengers. |
| SetupStage2 | AirLineTest | An object of type AirPlane with 3 passengers. They already are in the hash table.  **Passenger1:**(id="46514",name="Alejandro",miles=642,age=40,Plane=plane,row=4, seat=4)  **Passenger2:**(id="64844",name="Laura",miles=770,age=19,Plane=plane,row=8,seat=2)  **Passenger3:**(id="18070",name="Andres",miles=642,age=22,Plane=plane,row=9,seat=1) |
| SetupStage3 | AirLineTest | An object of type AirPlane with 4 passengers, 3 of them are in the hash table.  **Passengers registered in priority queue and Hash table:**  **Passenger1:**(id=“94864",name="Yeison",miles=645,age=19,Plane=plane,row=4,seat=4), entryOrder=1  **Passenger2:**(id=”65244",name="Daniel",miles=564,age=20,Plane=plane,row=8,seat=1), entryOrder=2  **Passenger3:**(id="94848","name=”Alejandra",miles=461,age=32,Plane=plane,row=8,seat=4), entryOrder=3  **Passenger register in hash table but not in priorityQueue:**  **Passenger4:**(id=”94474",name=”Cristian",miles=564,age=44,Plane=plane,row=7,seat=2) |
| SetupStage4 | AirLineTest | An object of type AirPlane with 4 passengers. all of them are in the hash table:  **Passenger1:**(id=”48654",name="Gustavo",miles=645,age=62,Plane=plane,row=4,seat=4), entryOrder=1.  **Passenger2:**(id=”84901",name="Federico",miles=564,age=45,Plane=plane,row=15,seat=1), entryOrder=3.  **Passenger3:**(id=”98614",name="Federico",miles=5135,age=22,Plane=plane,row=12,,seat=3), entryOrder=4.  **Passenger4:**(id=”59747",name="Margarita",miles=2568,,age=44,Plane=plane,row=12,seat=3), entryOrder=2. |

**Requerimiento 1: Load dataBase**

| **Test objective:** Verify that the LoadDataBase and loadPlane methods of the AirLine class work correctly. | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenery** | **Inputs** | **Expected result** |
| AirLine | **loadDataBase()** | SetupStage1 | A database with the next information:  (id::name::miles::age::row::seat)  13826::Santiago::4692::56::5::3  15348::Elena::2209::36::15::2  21734::Luis::3782::29::8::1  28963::Marco::3096::47::13::2  35691::Esteban::1189::31::15::1  36751::Maria::3433::55::17::3  41825::David::3953::39::10::2  47816::Sofia::3211::25::5::4  51327::Martina::3103::45::11::3  52479::Oscar::4288::50::3::2  57139::Lucia::4362::31::15::4  60583::Juan::3242::64::2::3  68043::Andrea::1318::22::1::1  73102::Mateo::4754::33::17::2  74512::Valeria::2556::57::16::3  80249::Ana::3155::49::11::4  88907::Jorge::2783::22::14::1  94628::Carla::4246::41::8::2  96275::Daniela::4834::39::2::4  99036::Camila::4266::28::11::2 | It is expected that the method deserializes all the information and puts the objects passenger in the hash table. |
| Airline | **loadPlane()** | SetupStage1 | A database with the next information:  (planeName::quantiyOfRows::chairsByRows::quantityOfFirstClassRows)  Barcino'sPlane::18::4::5 | It is expected that the method deserializes the txt and it creates the object plane. |

**Requirement 2: Register Passenger**.

| **Test objective:**  Verify that the methods that are needed to guarantee the method of registerPassenger() works correctly. | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenery** | **Inputs** | **Expected result** |
| AirLine | **registerPassenger()** | setupStage2 | **HTpassengers.insert:**((Passenger(id=”18070",name="Andres",miles=642,age=22,Plane=plane,row=20,seat=2)  (id=”18070”)) | It is expected that the passenger will not be added and the program throws an exception, because the passenger’s row is not in the plane **(Rows of plane=15)** |
| AirLine | **registerPassenger()** | setupStage2 | **HTpassengers.insert**((Passenger("id=01963",name="Natalia",miles=642,age=45,Plane=plane,row=4,seat=88)  (id=”01963”)) | It is expected that the passenger will not be added and the program throws an exception, because the passenger’s seat is not in the plane **(seatsByRows=4)** |
| AirLine | **registerPassenger()** | setupStage2 | **HTpassengers.search**("64844") | It is expected that the program can find the passenger “Laura” with all his features. |
| AirLine | **registerPassenger()** | setupStage2 | **HTpassengers.search**(80249“) | It is expected that the program throws an exception because there is not a passenger associated with that key. |
|  |  |  |  |  |

**Requirement 3:** **Show entry order**.

| **Test objective:** ShowEntryOrder(), is an output method. For that reason, the objective of the test is verifying that methods that are needed in the method showEntryOrder() work correctly. | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenery** | **Inputs** | **Expected result** |
| AirLine | **showEntryOrder()** | setupStage3 | **PQpassengers.extract().getName()** | It is expected that once the method is executed, it throws the name of the passenger “Yeison”, who has the most priority on stage. |
| AirLine | **ShowEntryOrder()** | setupStage3 | **PQpassengers.extract().getName()**  **PQpassengers.extract().getName()** | In this case, the priority of the other two passengers are the same (both are second class; what gives them the priority is the number their row in the plane). However in this case, it is expected that “Alejandra” will be the first who enters the plane, because she arrives first. |
| AirLine | **registerPassenger** | setupStage3 | *repeat:*  **PQpassengersIn.extract.getName()**  *until:*  **PQpassengersIn.getName.equals**(“Cristian”)  *end:*  **HTpassengers.search**(“94474”) | It is expected that the program shows that there is a passenger (Cristian) in the hash table but that passenger is not in the priority queue. |

**Requirement 4:** **Show out order.**

| **Test objective.** ShowOut() is an output. For that reason, the objective of this test is verifying that methods that are needed in the method showEntryOrder() work correctly. | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenery** | **Inputs** | **Expected result** |
| AirLine | **showOutOrder()** | setupStage4 | *repeat:*  **StackpassengerOut.push(PQpassengersOut.extract());**  *until:*  **PQpassengersOut.getHeapSize()==0**  *end:*  **airline.StackpassengerOut.pop().getName()** | It is expected that the program will pass all the information of the priority queue in the stack. For that reason it is expected that the program returns “Gustavo”, because he is the first person in the stack. |
| AirLine | **showOutOrder()** | setupStage4 | *repeat:*  **StackpassengerOut.push(PQpassengersOut.extract());**  *until:*  **PQpassengersOut.getHeapSize()==0**  *end:*  **airline.StackpassengerOut.pop().getName()**  **airline.StackpassengerOut.pop().getName()** | It is expected that once the program passes the information of the priority queue in the hash table, it returns “Margarita”, because the stack was popping twice and she arrives first than “Ana sofia”. |

**Test cases of data structures:**

**Stack:**

**\*For all the stages, exist a global object Plane, its features are the next:**

**(name=”Aereo”,rows=15,seatsByRows=4,firstClassRows=6”)**

| **Name** | **Class** | **Stage** |
| --- | --- | --- |
| SetupStage1 | StackTest | A generic stack without any passengers. |
| SetupStage2 | StackTest | A stack with 3 passengers:  **Passenger1:**(id="56453",name="Daron",miles=6511,age=18, Plane=plane,row=6,seat=2);  **Passenger2:**(id="98486",name="José",miles=4755,age=22,Plane=plane,row=2,seat=4);  **Passenger3:**(id="34636",name="Dayana",miles=7496,age=19,Plane=plane,row=12,seat=3) |

| **Test objective.** Validate that the methods of the data structure Stack works correctly. In the unexpected cases, the program will throw exceptions. | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenary** | **Inputs** | **Expected result** |
| Stack | **top()** | SetupStage2 | **stack.top().getName()** | It is expected that the program gets the name of the first passenger in the stack. The stacks is LIFO, for that’s reason, it is expected that returns “Dayana” |
| Stack | **push()** | SetupStage2 | **stack.push**((Passenger(id="65154",name="Mariana",miles=4985,age=20,Plane=plane,row=4,seat=2));  **stack.top().getName()** | It is expected that the program returns “Mariana” because she is the last passenger to be added to the stack. |
| Stack | **pop()** | SetupStage2 | **stack.pop()**  **stack.top.getName()** | It is expected that the program returns “Jose” because he is the second on the stack after “Dayana” |
| Stack | **isEmpty()** | SetupStage1 | **stack.isEmpty()** | It is expected that the program returns true, because the stack in the stage is empty. |
| Stack | **top()** | SetupStage1 | **stack.top()** | It is expected that the program launches an exception saying that the stack is empty. |
| Stack | **pop()** | SetupStage1 | **stack.pop()** | It is expected that the program launches an exception saying that the stack is empty. |

**Queue:**

**\*For all the stages, exist a global object Plane, its features are the next:**

**(name=”Aereo”,rows=15,seatsByRows=4,firstClassRows=6”)**

| **Name** | **Class** | **Stage** |
| --- | --- | --- |
| SetupStage1 | QueueTest | A generic queue without any passengers. |
| SetupStage2 | QueueTest | A stack with 3 passengers:  **Passenger1:**(id="56453",name="Javier",miles=5461,age=65,Plane=plane,row=10,seat=3)  **Passenger2:**(id=”98486", name="Lina",miles=4865,age=18,Plane=plane,row=1,seat=4)  **Passenger3:**(id="34636", name="Daniel",miles=1654,age=25,Plane=plane,row=3,seat=2) |

| **Test objective.** Validate that the methods of the data structure Queue works correctly. In the unexpected cases, the program will throw exceptions. | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Stage** | **Input values** | **Expected result** |
| Queue | **front()** | SetupStage2 | **stack.front().getName()** | It is expected that the program gets the name of the first passenger in the queue. The queue is FIFO, for that’s reason, it is expected that returns “Javier” |
| Queue | **enqueue()** | SetupStage2 | **queue.enqueue**((Passenger(id="65462",name="Manuel",miles=6555,age=47,Plane=plane,row=12,seat=4));  **queue.getLast()** | It is expected that the program returns “Manuel”, because he is the last added in the queue. |
| Queue | **dequeue()** | SetupStage2 | **queue.dequeue()**  **queue.front()** | It is expected that the program returns “Lina”, because she is the second person in the queue. |
| Queue | **isEmpty()** | SetupStage1 | **queue.isEmpty()** | It is expected that the program returns true, because in the stage, the queue is empty. |
| Queue | **dequeue()** | SetupStage1 | **queue.dequeue()** | It is expected that the program launches an exception because the queue is empty. |
| Queue | **front()** | SetupStage1 | **stack.front()** | It is expected that the program launches an exception because the queue is empty. |

**Hash table:**

**\*For all the stages, exist a global object Plane, its features are the next:**

**(name=”Aereo”,rows=15,seatsByRows=4,firstClassRows=6”)**

| **Name** | **Class** | **Stage** |
| --- | --- | --- |
| SetupStage1 | HashTableTest | A generic hash table without any passengers. |
| SetupStage2 | HashTableTest | A hash table with 3 passengers and their respectives keys:  **Passenger1:**(id="65684",name="Esteban",miles=1231,age=52,Plane=plane,row=10,seat=4) (id=”65684”)  **Passenger2:**(id=”84935",name="Felipe",miles=2686,age=16,Plane=plane,row=2,seat=1) (id=”84935”)  **Passenger3:**(id="49884",name="Sergio",miles=4636,age=43,Plane=plane,row=9,seat=2) (id=”49884”) |

| **Test objective.** Validate that the methods of the Hash table work correctly. In the unexpected cases, the program will throw exceptions. | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Stage** | **Input values** | **Expected result** |
| HashTable | **insert()** | SetupStage1 | **hashTable.insert**((Passenger:id="26566",name="David",miles=4656,age=70,Plane=plane,row=7,seat=3) (id=”26566))  **hashTable.search**(“26566”).**getName()** | It is expected that the program returns the name of the passenger inserted in the hash table (“David”). |
| HashTable | **search()** | SetupStage2 | **hashTable.search**(“65684”)**.getName()** | It is expected that the program returns “Esteban”, because he is the passenger associated with that key. |
| HashTable | **search()** | SetupStage2 | **hashTable.search**(“53464”) | It is expected that the program throws an exception, because there is not a passenger associated with that key. |

**Heap:**

**\*For all the stages, exist a global object Plane, its features are the next:**

**(name=”Aereo”,rows=15,seatsByRows=4,firstClassRows=6”)**

| **Name** | **Class** | **Stage** |
| --- | --- | --- |
| SetupStage1 | HeapTest | A generic heap without any passengers. |
| SetupStage2 | HeapTest | Heap with 3 passengers with their priority.  **Passenger1:**(id="65684",name="Esteban",miles=1231,age=52,Plane=plane,row=10,seat=4) (priority=15)  **Passenger2:**(id=”84935",name="Felipe",miles=2686,age=16,Plane=plane,row=2,seat=1) (priority=12)  **Passenger3:**(id="49884",name="Sergio",miles=4636,age=43,Plane=plane,row=9,seat=2) (priority=20) |

| **Test objective.** Validate that the methods of the heap work correctly. In the unexpected cases, the program will throw exceptions. | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Stage** | **Input values** | **Expected result** |
| Heap | **insertPassenger()** | SetupStage1 | **heap.insertPassenger**((Passenger:id="56540",name="Mauricio",miles=5464,age=40,Plane=plane,row=7,seat=3)) (prirority=18) | It is expected that the passenger will be inserted in the heap and that the heap size will be updated to one. |
| Heap | **getRoot()** | SetupStage2 | **heap.getRoot().getName()** | It is expected that the program returns “Facundo”, because he is the passenger with the most priority. |
| Heap | **getRoot()** | SetupStage1 | **heap.getRoot()** | It is expected that the program throws an exception, because there is not a root in the stage. |
| Heap | **extract()** | setupStage2 | **heap.extract().getName()** | It is expected that the program returns “Facundo” because he has the most priority. |
| Heap | **extract** | SetupStage1 | **heap.extract()** | It is expected that the program throws an exception because there is no root in the heap. |

**Temporal and spatial complexity of the implemented methods:**

| **public void loadDataBase() throws Exception** | | **Size in memory** | **Quantity (Size memory)** | **Temporal**  **Complexity** |
| --- | --- | --- | --- | --- |
| **1** | *loadDataBase=true;* | **16 bits** | **1** | **1** |
| **2** | *File file = new File(path);* | **64 bits** | **1** | **1** |
| **3** | *FileInputStream fis = new FileInputStream(file);* | **64 bits** | **1** | **1** |
| **4** | *BufferedReader reader = new BufferedReader(new InputStreamReader(fis));* | **8KB =**  **65,536 bits** | **1** | **1** |
| **5** | *String line = "";* | **40 bytes =**  **320 bits** | **1** | **1** |
| **6** | *String[] arr;* | **64 bits** |  | **1** |
| **7** | *String content = "";* | **40 bytes =**  **320 bits** | **1** | **1** |
| **8** | *while ((line = reader.readLine()) != null) {* | **107 bytes =**  **856 bits** | **n+1** | **n+1** |
| **9** | *arr = line.split("::");* | **328 bytes =**  **2624 bits** | **n** | **n** |
| **10** | *if (Integer.parseInt(arr[4]) <= thePlane.getRows()&&Integer.parseInt(arr[5])<= thePlane.getChairsByRows()) {* | **0** | **n** | **n** |
| **11** | *HTpassengers.insert(*  *arr[0], new Passenger(arr[0], arr[1], Integer.parseInt(arr[2]), Integer.parseInt(arr[3]), thePlane, Integer.parseInt(arr[4]),Integer.parseInt(arr[5])));*  *}*  *}*  } | **32 bytes=**  **256 bits** | **n** | **n** |
|  | **Total:** | **69928 bits =**  **8741 bytes** | **4n+7** | **4n +7 = O(n)** |

| **public void showEntry() throws Exception {** | | **Size in memory** | **Quantity (Size memory)** | **Temporal**  **Complexity** |
| --- | --- | --- | --- | --- |
| **1** | *int counter=0;* | ***16 bits*** | **1** | **1** |
| **2** | *while(PQpassengersIn.getHeapSize()!= 0&&thePlane.getSeatsByRows()\* thePlane.getRows()>counter) {* | **107 bytes =**  **856 bits** | **n+1** | **n+1** |
| **3** | *counter++;* | **32 bits** | **n** | **n** |
| **4** | *EntryOrder=EntryOrder+PQpassengersIn.extract().toString()+"\n";*  *}*  *}* | **8KB =**  **64,542 bits** | **n** | **n** |
|  | **Total:** | **65,590 bits =**  **8073 bytes** | **3n+2 =O(n)** | **3n+2 =O(n)** |