**ENGINEERING METHOD**

**JUAN FELIPE JOJOA CRESPO A00382042**

**FELIPE ROJAS PRADO A00393918**

**Phase 1: Problem Understanding**

**Problem Description:**

A first version of a system for an airline is required to improve the process of boarding and exiting the plane. Specifically, the system is expected to:

* Load passenger information corresponding to a flight from a plain text file.
* Efficiently search for passenger information.
* Record the arrival of a passenger to the boarding room.
* Show the order in which passengers should board the plane, considering the order of arrival and the sections of the plane.
* Establish special rules for the boarding of first-class passengers, taking into account their order of arrival, accumulated miles, special attention required, old age, or other relevant data.

**Functional Requirements:**

**RF1: Upload passenger information:** The system must allow loading passenger information corresponding to a flight through a plain text file generated by users.

**RF2: Record the arrival of passengers:** Once passengers arrive at the corresponding boarding room, the system must allow for efficient searching of their complete information and record their arrival to the room.

**RF3: Efficient search of the complete information of a passenger in the boarding room:** The system must be capable of finding a passenger's information in the shortest possible time, once the passenger has arrived at the gate. The search for complete passenger information may include data such as first name, last name, flight number, seat number, among others.

**RF4: Show order of boarding the plane:** The system must allow the crew member in charge to show in which order passengers should board the plane, considering the order of arrival and the section of the plane being called.

**RF5: Special rules for first class:** The system must allow for special rules to be applied for first-class passengers, taking into account other relevant data such as accumulated miles, special attention required, old age, or other relevant data.

**RF6: Show the order of departure of passengers from the plane:** The system must allow the crew member in charge to show in which order passengers should leave the plane, considering the special rules mentioned above.

**Non-functional Requirements:**

**RNF 1:** **Efficiency in the search for information:** The system must be efficient in the search for passenger information to register their arrival at the boarding gate.

**RNF2: Scalability:** The system must be able to handle large amounts of data in the future, without affecting performance.

**RNF 3:** **Usability:** The system must be easy to use for the crew member in charge, with an intuitive and clear interface.

**RNF4: Security:** The system must guarantee the security of passenger data, avoiding unauthorized access or information leaks.

**RNF5: Availability:** The system must be available at all times, without interruptions, to guarantee the correct process of entry and exit of passengers.

**Restrictions:**

* You must use your own data structures (predefined libraries cannot be used).
* The privacy of passenger information must be guaranteed.
* The system must be developed in Java.

**REQUIREMENTS TABLE INTEGRATIVE TASK CYED 1**

| Client | Airline |
| --- | --- |
| User | Airline staff, Passengers |
| Functional requirements | * **R1:** Upload passenger information. * **R2:** Register the arrival of passengers. * **R3:** Efficient search of the complete information of a passenger in the boarding room. * **R4:** Show order of entry to the plane. * **R5:** Special rules for the first class. * **R6:** Show the order of departure of the passengers from the plane. |
| Problem context | A renowned airline has selected a group of students with excellent academic performance to develop a first version of a system that improves order in the process of entering and exiting passengers on planes. For the first version of the system, students are expected to simulate the passenger database using a plain text file and the system will allow the loading of passenger information for a given flight.  The system must allow efficient search of the complete information of a passenger once they arrive at the corresponding boarding gate, and record their arrival at the gate to reward the punctuality of the passengers entering the plane in the order of arrival. In addition, the system must allow the crew member in charge to be shown in which order the passengers should enter, taking into account the special rules that apply to first class passengers. Finally, the system must establish an exit order for passengers that takes into account row configuration and proximity to the aisle, and allow the flight crew person in charge of the system to see in which order they should exit. the passengers. |
| Non-functional requirements | * **RNF1:** Efficiency in the search for information: The system must be efficient in the search for passenger information to register their arrival at the boarding gate. * **RNF2:** Scalability: The system must be able to handle large amounts of data in the future, without affecting performance. * **RNF3:** Usability: The system must be easy to use for the crew member in charge, with an intuitive and clear interface. * **RNF4:** Security: The system must guarantee the security of passenger data, avoiding unauthorized access or information leaks. * **RNF5:** Availability: The system must be available at all times, without interruptions, to guarantee the correct process of entry and exit of passengers. |

| Name or identifier | **R1** | | |
| --- | --- | --- | --- |
| Abstract | The system must allow the loading of passenger information corresponding to a flight by means of a plain text file generated by the users. | | |
| Inputs | Input name | Data type | Condition of select or repetition |
| filePath | String, CSV | Can’t be null |
| name | String | Can’t be null |
| seatNumber | String | Can’t be null |
| ticket | String | Can’t be null |
| age | Integer | Can’t be null or less than 0 |
| accumulated Miles | Integer | Can’t be null or less than 0 |
| specialAttention | Boolean | Can’t be null |
| date | Date |  |
| General activities needed to obtain the results | 1. Access the system. 2. Access the upload passenger information module. 3. Select the flight for which to upload passenger information. 4. Browse and select the CSV file containing the passenger information. 5. Validate the file format and structure. 6. Map the columns in the CSV file to the corresponding fields in the passenger information database. 7. Upload the passenger information to the system. | | |
| Result or postcondition | The passenger information for the selected flight is successfully uploaded to the system and available for subsequent processes. | | |
| Outputs | Output name | Data type | Condition of select or repetition |
| MsgConfirmation | String |  |

| Name or identifier | **R2** | | |
| --- | --- | --- | --- |
| Abstract | The system must be able to register the arrival of passengers. | | |
| Inputs | Input name | Data type | Condition of select or repetition |
| filePath | String, CSV | Can’t be null |
| flightNumber | String | Can’t be null |
| name | String | Can’t be null |
| seatNumber | String | Can’t be null |
| ticket | String | Can’t be null |
| age | Integer | Can’t be null or less than 0 |
| accumulated Miles | Integer | Can’t be null or less than 0 |
| specialAttention | Boolean | Can’t be null |
| date | Date |  |
| General activities needed to obtain the results | The system searches for the complete information of the passenger in the boarding room and registers their arrival. | | |
| Result or postcondition | The passenger's arrival is successfully registered in the system. | | |
| Outputs | Output name | Data type | Condition of select or repetition |
| MsgConfirmationArrival | String |  |

| Name or identifier | **R3** | | |
| --- | --- | --- | --- |
| Abstract | The system must be capable of finding a passenger's information in the shortest possible time, once the passenger has arrived at the gate. The search for complete passenger information may include data such as name, flight number, seat number, among others. | | |
| Inputs | Input name | Data type | Condition of select or repetition |
| filePath | String, CSV | Can’t be null |
| flightNumber | String | Can’t be null |
| name | String | Can’t be null |
| seatNumber | String | Can’t be null |
| ticket | String | Can’t be null |
| age | Integer | Can’t be null or less than 0 |
| accumulated Miles | Integer | Can’t be null or less than 0 |
| specialAttention | Boolean | Can’t be null |
| date | Date |  |
| General activities needed to obtain the results | 1. Receive passenger's identifying information. 2. Search the database for the passenger's complete information. 3. Retrieve the complete information of the passenger. 4. Display the complete information of the passenger. | | |
| Result or postcondition | The complete information of the passenger is displayed. | | |
| Outputs | Output name | Data type | Condition of select or repetition |
| date | Date | Can’t be null |
| MsgFoundedPassenger | String |  |

| Name or identifier | **R4** | | |
| --- | --- | --- | --- |
| Abstract | Show the order in which passengers should board the plane. | | |
| Inputs | Input Name | Data Type | Condition of select or repetition |
| boardingOrder | Integer | Can’t be null or less than 0 |
| boardingSectionsList | String | Can’t be null  The passenger information must be available in the system before boarding starts. |
| filePath | String, CSV | Can’t be null |
| flightNumber | String | Can’t be null |
| name | String | Can’t be null |
| seatNumber | String | Can’t be null |
| ticket | String | Can’t be null |
| age | Integer | Can’t be null or less than 0 |
| accumulated Miles | Integer | Can’t be null or less than 0 |
| specialAttention | Boolean | Can’t be null |
| date | Date |  |
| General activities needed to obtain the results | 1. Retrieve passenger information from the system. 2. Sort passengers by boarding section and arrival order. 3. Display the order of boarding on the screen or on printed boarding passes. | | |
| Result or postcondition | Passengers board the plane in the correct order, ensuring a smooth and efficient boarding process. | | |
| Outputs | Output name | Data type | Condition of select or repetition |
| boardingOrderList | String | The boarding order must be clear and easy to understand for the flight crew and passengers. |
| updatedPassenger | String |

| Name or identifier | **R5** | | |
| --- | --- | --- | --- |
| Abstract | The system must apply special rules for the boarding of first-class passengers, considering factors such as accumulated miles, special attention required, elderly passengers, or other relevant data. | | |
| Inputs | Input Name | Data Type | Condition of select or repetition |
| filePath | String, CSV | Can’t be null |
| flightNumber | String | Can’t be null |
| name | String | Can’t be null |
| seatNumber | String | Can’t be null |
| ticket | String | Can’t be null |
| age | Integer | Can’t be null or less than 0 |
| accumulated Miles | Integer | Can’t be null or less than 0 |
| specialAttention | Boolean | Can’t be null |
| date | Date |  |
| General activities needed to obtain the results | 1. Load passenger information and special rules into the system. 2. Apply the special rules to determine the boarding order for the first-class passengers. | | |
| Result or postcondition | The system must determine the order of boarding for first-class passengers based on special rules. | | |
| Outputs | Output name | Data type | Condition of select or repetition |
| boardingOrderList | String, List | The boarding order must be clear and easy to understand for the flight crew and passengers. |
|  |  |

| Name or identifier | **R6** | | |
| --- | --- | --- | --- |
| Abstract | The system must display the order in which passengers should leave the plane, taking into account the special rules mentioned in the previous requirement. | | |
| Inputs | Input Name | Data Type | Condition of select or repetition |
| None. | | |
|
|
|
|
|
|
|
|
| General activities needed to obtain the results | 1. Retrieve the passenger information from the system. 2. Sort the passengers based on the special rules for first-class passengers. 3. Sort the remaining passengers based on their seat location or any other relevant criteria. | | |
| Result or postcondition | The system displays the order in which passengers should leave the plane, taking into account the special rules for first-class passengers. | | |
| Outputs | Output name | Data type | Condition of select or repetition |
| orderDepartureList | String, List | The departure order must be clear and easy to understand for the flight crew and passengers. |
|  |  |

**Phase 2:**

* **Load txt File and GSON Save:** Data load txt and save Gson are two commonly used methods for storing and retrieving data in software applications.

Data load txt refers to the process of loading data from a text file into an application. The text file typically contains data in a structured format, such as comma-separated values or JSON, which can be easily parsed and loaded into the application.

On the other hand, save Gson refers to the process of converting an object or collection of objects into JSON format and then saving it to a file or database. Gson is a popular Java library that provides easy-to-use methods for converting objects to JSON and vice versa.

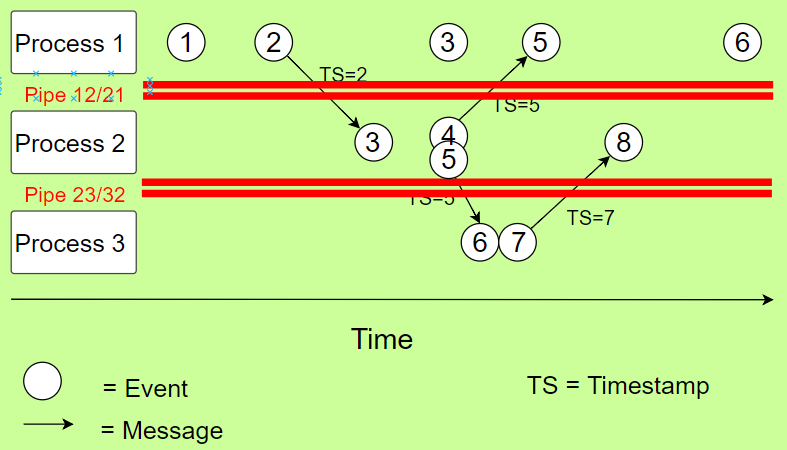


**Why would we use it?:** Regarding the implementation of a hash table in the given problem, we could use the data load txt method to load data from a text file containing passenger information into a hash table. Each passenger could be represented as an object with attributes such as name, arrival time, departure time, etc. We could use a unique identifier, such as the passenger's name or a generated ID, as the key for the hash table, allowing for efficient retrieval of passenger information.

Similarly, we could use the save Gson method to serialize the passenger objects in the hash table to JSON format and save them to a file or database for future use. This would allow us to easily store and retrieve passenger information in a structured format that can be easily parsed and manipulated by other software components.

**Bibliographic reference:** [**https://www.baeldung.com/gson-save-file**](https://www.baeldung.com/gson-save-file)

* **Data and Time functions:** In computer science, a timestamp is a sequence of characters or encoded information that indicates when an event occurred, usually representing a specific date and time. It's commonly used for logging events or keeping track of when particular operations happened.



**Why would we use it?:**

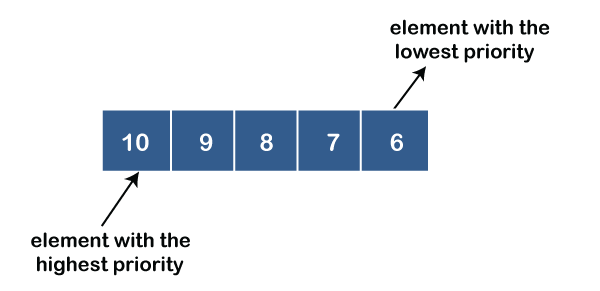
In the context of the problem statement, timestamps could be used to record the time when a passenger arrives at the station and when they board the train. This information can be stored along with the passenger's ID and other relevant details to keep track of the movement of passengers through the system.

Regarding the implementation of a hash table, timestamps can be used as keys to uniquely identify each passenger's record. The hash function could be designed to generate a hash value based on the timestamp and use it to index the corresponding record in the hash table. This would allow for efficient retrieval and manipulation of passenger data based on their timestamp.

**Bibliographic reference:** [**https://ieeexplore.ieee.org/document/805196**](https://ieeexplore.ieee.org/document/805196)

* **Priority Queue:** A priority queue is a data structure that stores elements with associated priorities and allows efficient access to the element with the highest priority. In other words, it is similar to a regular queue, but each element has a priority assigned to it and elements with higher priority are dequeued before elements with lower priority.

One way to implement a priority queue is by using a heap data structure, such as a binary heap. Another approach is to use a balanced binary search tree, such as a red-black tree or AVL tree.



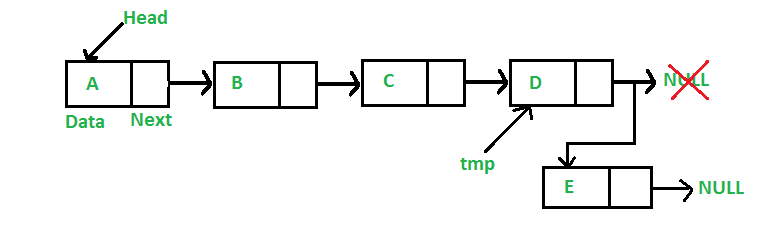
**Why would we use it?:** In the context of the airline boarding problem described in the prompt, a priority queue could be used to keep track of the order in which passengers should board the plane. Each passenger would have a priority assigned to them based on their boarding group (e.g., first class, business class, economy class), their position within that group (e.g., based on miles accumulated or special needs), and their arrival time. The priority queue would allow the system to efficiently dequeue the next passenger to board based on their priority.

As for how a hash table could be implemented in this problem, it could be used to efficiently lookup passenger information based on their name or other unique identifier. The hash table would store key-value pairs, where the keys are the passenger names or IDs and the values are the corresponding passenger objects. This would allow the system to quickly retrieve a passenger's information when they arrive at the boarding gate and need to be checked in.

**Bibliographic reference:** [**https://www.geeksforgeeks.org/priority-queue-set-1-introduction/**](https://www.geeksforgeeks.org/priority-queue-set-1-introduction/)

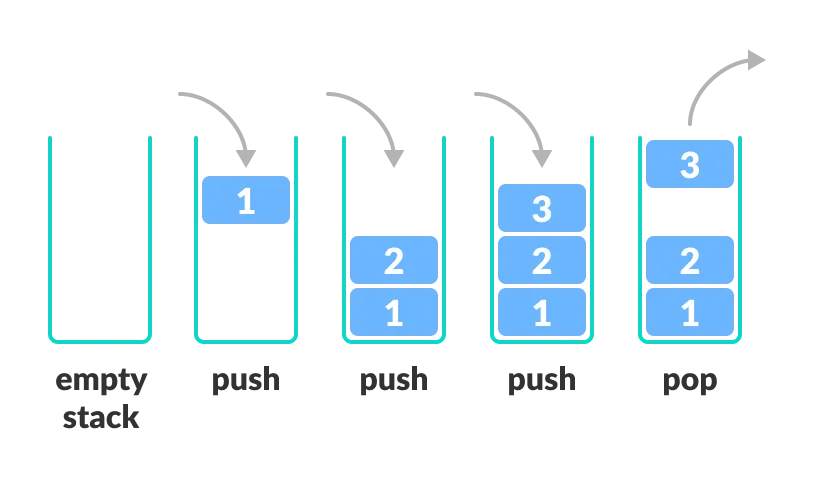
* **Linked List:** A linked list is a data structure that represents a sequence of elements, where each element points to the next element in the sequence through a pointer or reference. It consists of a series of nodes, where each node contains a value or data and a reference to the next node in the sequence.

To implement a hash table, you could use linked lists to handle collisions. When two keys map to the same index in the hash table, instead of overwriting the previous value, you can add the new value to the linked list at that index. When searching for a specific key, you can first find the correct index in the hash table and then iterate through the linked list at that index until you find the key you are looking for.



**Bibliographic reference:** [**https://www.geeksforgeeks.org/what-is-linked-list/**](https://www.geeksforgeeks.org/what-is-linked-list/)

* **Stack:** A stack is a linear data structure in which elements are added and removed from only one end, called the top. It follows the Last-In-First-Out principle, meaning that the last element added to the stack is the first one to be removed. Stacks are commonly used in computer science, for example in programming languages for managing function calls and in undo-redo operations.



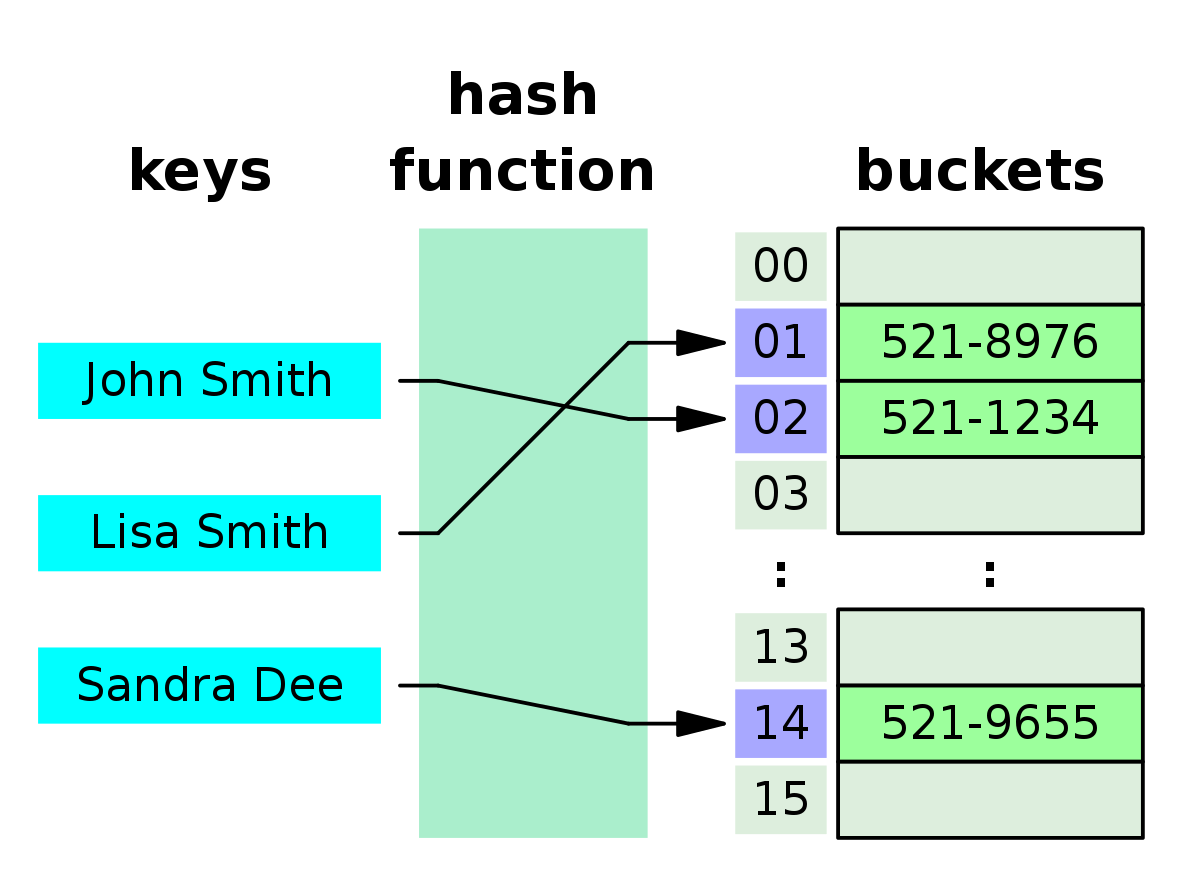
**Why would we use it?:**

In the context of the airline system described in the prompt, a stack could be used to keep track of the order in which passengers should board the plane. Each time a passenger arrives at the boarding gate, their ID could be added to the top of the stack, indicating that they are the most recent passenger to arrive. When it's time for boarding, the passenger IDs can be removed from the stack in reverse order, with the last passenger added being the first to board the plane.

However, a hash table could also be used in conjunction with the stack to allow for prioritization of certain passengers, as mentioned in my previous answer. The hash table could store the priority score for each passenger, with the passenger ID as the key and the score as the value. When a passenger arrives at the gate, their ID and priority score could be added to both the hash table and the stack. When it's time for boarding, the stack can be emptied in reverse order of priority score by first retrieving the IDs of passengers with the highest priority score from the hash table, then removing them from the stack in that order. This would ensure that passengers are boarded in order of priority, with first class passengers or those with special needs being given priority over other passengers.

**Bibliographic reference:** [**https://www.geeksforgeeks.org/introduction-to-stack-data-structure-and-algorithm-tutorials/**](https://www.geeksforgeeks.org/introduction-to-stack-data-structure-and-algorithm-tutorials/)

* **Hash Table:** A hash table is a data structure that allows efficient retrieval and storage of key-value pairs. It works by using a hash function to map a given key to a specific index in an array where the corresponding value can be stored. The hash function takes the key as input and returns a numerical value which is used as the index in the array. This allows for fast access to the value associated with a given key.



**Why would we use it?:**

In the context of the airline system described in the prompt, a hash table could be

used to efficiently store and retrieve passenger data based on their unique identifiers

(such as a passenger ID or passport number). The passenger ID could be used as

the key in the hash table, with the corresponding value being the passenger's data

(such as their name, seat number, special needs, etc.). This would allow for fast and

efficient retrieval of a passenger's data when they arrive at the boarding gate or when

it's time for them to exit the plane.

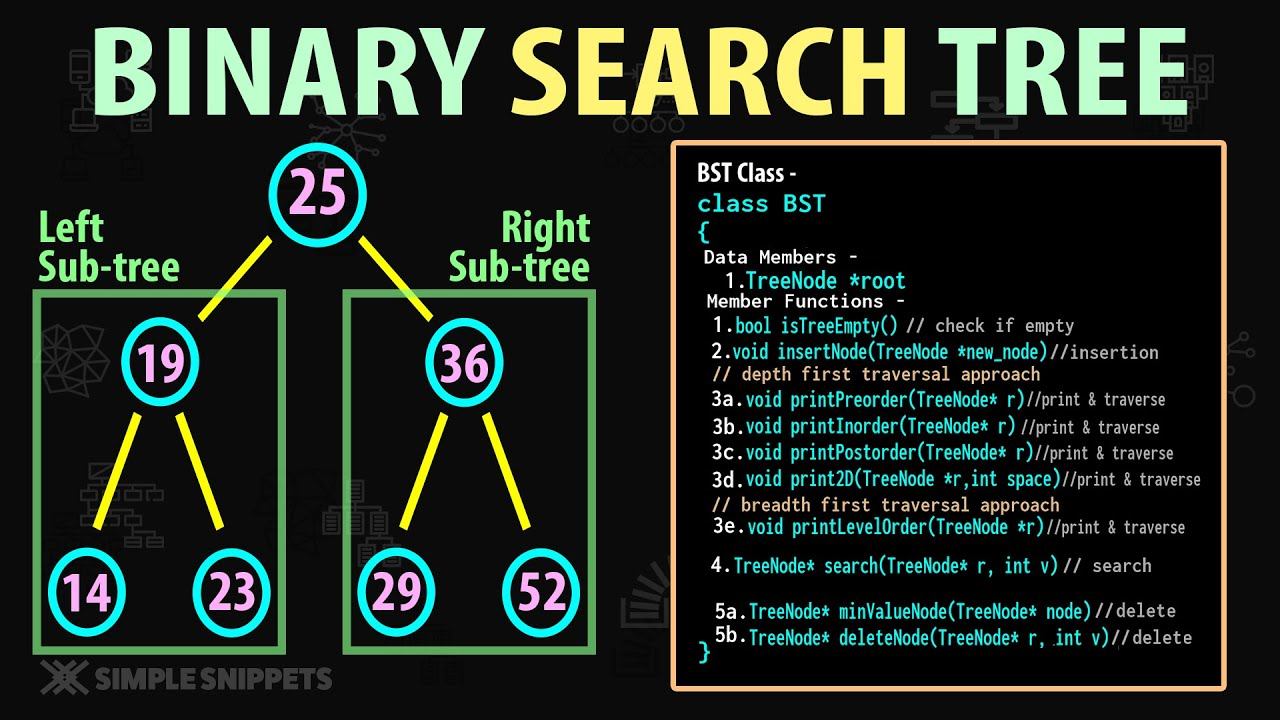
Additionally, the hash table could be extended to allow for prioritization of certain passengers, such as first class passengers. The hash function could be modified to take into account additional criteria, such as the number of miles the passenger has accumulated or their age, in order to assign a priority score to each passenger. This score could then be used as the index in the hash table, allowing for easy sorting and retrieval of passengers in order of priority.

In the context of the airline system described in the prompt, a hash table could be used to efficiently store and retrieve passenger data based on their unique identifiers (such as a passenger ID or passport number). The passenger ID could be used as the key in the hash table, with the corresponding value being the passenger's data (such as their name, seat number, special needs, etc.). This would allow for fast and efficient retrieval of a passenger's data when they arrive at the boarding gate or when it's time for them to exit the plane.

Additionally, the hash table could be extended to allow for prioritization of certain passengers, such as first class passengers. The hash function could be modified to take into account additional criteria, such as the number of miles the passenger has accumulated or their age, in order to assign a priority score to each passenger. This score could then be used as the index in the hash table, allowing for easy sorting and retrieval of passengers in order of priority.

**Bibliographic reference:** [**https://cs.stackexchange.com/questions/106740/implementing-a-symbol-table-with-a-hash-table-and-a-stack**](https://cs.stackexchange.com/questions/106740/implementing-a-symbol-table-with-a-hash-table-and-a-stack)

* **Binary Search Tree:** A binary search tree is a data structure in computer science used for organizing and storing data. It is a tree-like structure in which each node has at most two children, referred to as the left child and the right child. The left child node contains data that is less than the parent node, while the right child node contains data that is greater than the parent node. This organization of nodes allows for efficient searching, insertion, and deletion of data.



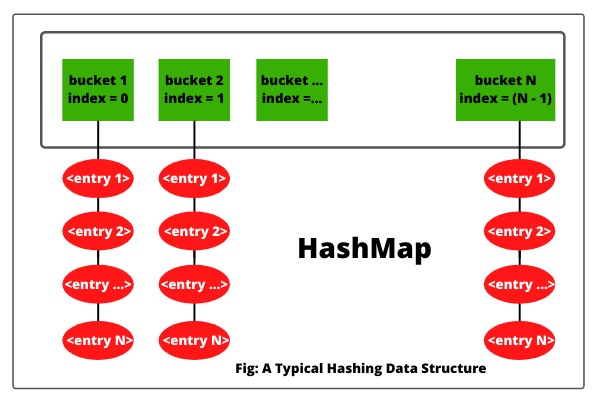
**Why would we use it?:** a BST could be used to store and look up flight information in an ordered manner, for example by flight number or departure date.

In this way, when an information query for a particular flight is received, it can quickly search the binary search tree and obtain the requested information in an efficient time. The BST could also be used to store and look up passenger information, such as their identification number or assigned seat number.

**Bibliographic reference:**

<https://www.geeksforgeeks.org/binary-search-tree-data-structure/>

* **Hashmap:** A HashMap is a data structure used to efficiently store and access data. It works by using keys and values, where each key is assigned a specific value. This allows data to be searched and accessed quickly using the corresponding key.

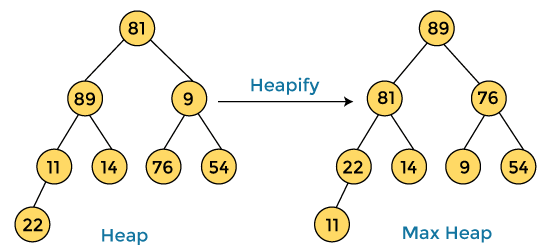


For example, if we have a list of passengers on a flight, we can use a HashMap to assign each passenger a seat number. This would allow us to quickly search for a passenger's information in the database using their seat number.

**Why would we use it?:** This data structure can be useful for looking up a passenger's information by means of a specific key, such as seat number. This can improve database lookup time and increase overall system efficiency.

**Bibliographic reference:** [**https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html**](https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html)

* **Heapsort:** Heapsort is a sorting algorithm that uses a binary heap data structure to sort elements. The basic idea is to convert the input array into a binary heap, which has the property that the parent node is always greater than its children nodes (max heap). Then, the maximum element is repeatedly removed from the heap and placed at the end of the array, which results in a sorted array in ascending order.



**Bibliographic reference:** <https://www.geeksforgeeks.org/heap-sort/>

**Phase 3: Proposed Solutions**

We have to know that proposing solution options for problems serves to identify and address the obstacles or challenges that arise in a certain context. This can help improve efficiency, reduce costs, increase productivity, improve quality, and ultimately achieve desired goals.

In addition, the solution proposal can also encourage creativity and critical thinking in the people involved, which can lead to significant improvements in the process and greater customer or user satisfaction.

In summary, proposing solution options is an important part of the problem solving process and can lead to significant improvements in the efficiency and effectiveness of any process or situation.

The solutions for improving the airline procedure are divided into different types, including Load and Save, Database, Sort, and Date. Depending on specific situations that may arise during the procedure, there are multiple solutions available for each type.

1. **Load and Save**

* Simple text File
* Excel file .csv
* Json in text file

1. **Database**

* Binary Search Tree
* HashMap
* HashTable
* Stack
* Priority Queue
* Arrays
* ArrayList

1. **Sort**

* HeapSort
* BubbleSort
* CollectionsSort
* SelectionSort
* InsertionSort

1. **Date**

* Actual Time Function
* Calendar Function
* Date format Function

**Phase 4: Discarded Solutions**

1. *Load and save:*

* **Excel file:** We assume that this type of format is not valid according to the requirements obtained, we are interested in the simple text file that the initial statement asks us for.

1. *Database:*

* **ArrayList:** Seeing the needs of the requirements, we do not necessarily need a List array since the number of seats is already configured in the flight by default.
* **Binary Search Tree:** We discard the idea of ​​using a binary search tree since it consists of having parent and child nodes, which does not fit the initial purpose of the exercise, which is to organize the passengers in the pre-established seats.

1. *Sort:*

* **Selection sort, insertion sort and BubbleSort:** According to the requirements of the statement that we managed to get, we are asked that the ordering algorithm must be efficient and capable as a non-functional requirement of being able to organize large numbers of people to be organized in airplane seats, we must also get the algorithm with as much complexity efficiency as possible.

1. *Date:*

* **Date format function:** According to the requirements of the statement that we were able to obtain, it is not appropriate to use this type of format in the implementation of the problem, since we need complete information on the arrival time of the plane passengers.

**Phase 5: Evaluation Criteria:**

In order to determine the optimal solution for implementation, we will develop an evaluation system using String values to simplify the assessment process.

**Evaluation System:**

A = Excellent

B = Good

C = Average

D = Below average

F = Failure or insufficient

* U: Usability
* M: Maintainable
* S: Scalability
* E: Efficiency

*Load and save*

* Text Simple file:

U) B

M) B

S) C

E) F

* Json Text File:

U) A

M) B

S) C

E) B

*Database*

* HashTable:

U) B

M) B

S) C

E) A

* Array:

U) D

M) A

S) F

E) F

* Stack:

U) C

M) B

S) B

E) B

* Priority Queue:

U) C

M) B

S) B

E) B

* Stack:

U) C

M) B

S) B

E) B

*Sort:*

* Heapsort:

U) B

M) B

S) A

E) A

* Collections Sort:

U) D

M) B

S) B

E) D

*Date*

* Current Time:

U) A

M) B

S) B

E) C

* Calendar Time:

U) A

M) B

S) B

E) C

After analyzing the evaluation criteria, we have concluded that the following functions are the most suitable:

* Hash Tables for their efficiency and scalability, as well as their ability to easily access and store information. This is in contrast to fixed arrays, queues, and stacks which lack direct access to stored data.
* Json for loading and saving data due to its user-friendly and familiar nature when compared to txt files, which would require learning new functions.
* Heapsort for its efficiency and maintainability compared to collection sort, which would require a comparator function and sorting each time.
* Calendar for its familiarity, even though it is as good as the current time function.

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

**Algorithm pseudocode:**

Algoritmo OrdenIngresoSalidaAvion

Definir archivo Pasajeros como Archivo de Texto

Definir lista Pasajeros como Lista de Registros

Definir lista PasajerosPrimeraClase como Lista de Registros

Definir entero NumeroFilasAvion

Definir entero NumeroColumnasAvion

Definir matriz AsientosAvion como matriz de enteros

Inicio

Abrir Pasajeros para lectura

Si Pasajeros Existe Entonces

Mientras No fin(Pasajeros) Hacer

Leer Pasajeros, RegistroActual

Si RegistroActual es válido Entonces

Si RegistroActual es de Primera Clase Entonces

Agregar RegistroActual a PasajerosPrimeraClase

Sino

Agregar RegistroActual a Pasajeros

Fin Si

Fin Si

Fin Mientras

Cerrar Pasajeros

Fin Si

Ordenar lista Pasajeros por campo OrdenLlegada

Ordenar lista PasajerosPrimeraClase por campo OrdenLlegada

Funcion BuscarPasajero(DatoBusqueda) como Registro

Para cada Pasajero en Pasajeros hacer

Si Pasajero.DatoBusqueda = DatoBusqueda Entonces

Devolver Pasajero

Fin Si

Fin Para

Para cada Pasajero en PasajerosPrimeraClase hacer

Si Pasajero.DatoBusqueda = DatoBusqueda Entonces

Devolver Pasajero

Fin Si

Fin Para

Devolver Registro Vacío

Fin Funcion

Para i=1 Hasta NumeroFilasAvion hacer

Para j=1 Hasta NumeroColumnasAvion hacer

Si AsientosAvion[i][j] <> 0 Entonces

PasajeroActual = BuscarPasajero(AsientosAvion[i][j])

Mostrar "El pasajero " + PasajeroActual.Nombre + " " + PasajeroActual.Apellido + " ingresó al avión en el asiento " + i + "-" + j

Fin Si

Fin Para

Fin Para

Para i=1 Hasta NumeroFilasAvion hacer

Para j=1 Hasta NumeroColumnasAvion hacer

Si AsientosAvion[i][j] <> 0 Entonces

PasajeroActual = BuscarPasajero(AsientosAvion[i][j])

Mostrar "El pasajero " + PasajeroActual.Nombre + " " + PasajeroActual.Apellido + " salió del avión desde el asiento " + i + "-" + j

Fin Si

Fin Para

Fin Para

Fin

FinAlgoritmo