## Enunciado:

https://docs.google.com/document/d/1zhlTbfgtojD6XqrFrqk5SPdd2NDFzDNMZAduw0jlj5M/edit

**rubrica:**
https://docs.google.com/spreadsheets/d/1C_NV7rlemvfb1_tuywoN4cZSab5RCb1o7ERHs9SrzWk/edit#gid=1014352675

### Entregables

1. Desarrollo completo del Método de la Ingeniería. **(Ejemplo)**
    a. La fase 1 debe incluir la especificación de requerimientos.

### Requerimientos

| | |
|---|---|
| **Cliente** | A recognized airline |
| **Usuario** | Tripulation and the airline team |
| **Requerimientos funcionales** | **R1: Load passenger info**<br>**R2: Search a passenger info**<br>**R3: Register the arrival to the gate**<br>**R4: Show boarding order**<br>**R5: Show unboarding order** |
| **Requerimientos no funcionales** | Efficiency in the passenger search<br><br>Use of correct data structures for handling passengers and their order of entry and exit from the plane.<br><br>Easy to use for the crew member in charge of the system.<br><br>Security in handling passenger information. |
| **Contexto del problema** | Una reconocida aerolínea quiere desarrollar un programa para realizar una primera versión de un sistema cuyo objetivo principal es mejorar el orden en el proceso de ingreso y salida del avión.<br>A recognized airline wants to develop a program to carry out a first version of a system whose main objective is to improve the order in the boarding and unboarding process of the plain |

| | | | |
|---|---|---|---|
| **Nombre o identificador** | **R1: Load passenger info** | | |
| **Resumen** | given an airline flight, the system must allow the loading of passenger information for that flight from a plain text document | | |
| **Entradas** | **Nombre de entrada** | **Tipo de Dato** | **Condición de selección o repetición** |
| | data | TXT | valid data |
| | | | |
| **Actividades generales necesarias para obtener los resultados** | 1. Enter the System<br>2. Choose one option<br>3. input file | | |
| **Resultado o Postcondición** | A confirmation message when the data is uploaded. | | |
| **Salidas** | **Nombre salida** | **Tipo de dato** | **Condición de selección o repetición** |
| | msg | String | |

| | | | |
|---|---|---|---|
| **Nombre o identificador** | **R2: Search a passenger info** | | |
| **Resumen** | When a passenger arrives at the boarding gate to wait for their flight, their complete information must be searched in the most efficient way possible (since the amount of data will be significantly large in the future). | | |
| **Entradas** | **Nombre de entrada** | **Tipo de Dato** | **Condición de selección o repetición** |
| | passengerName | String | |

| | | | |
|---|---|---|---|
| **Actividades generales necesarias para obtener los resultados** | The system has to search for a passenger in the database with the same name and if found, return all the information for that passenger | | |
| **Resultado o Postcondición** | if found, the System will return the info corresponding to the searched Passenger. Else, an error message. | | |
| **Salidas** | **Nombre salida** | **Tipo de dato** | **Condición de selección o repetición** |
| | passenger | String | |

| | | | |
|---|---|---|---|
| **Nombre o identificador** | **R3: Register the arrival to the gate** | | |
| **Resumen** | Once a passenger arrives at the corresponding boarding gate to wait for their flight, after searching for their information, they must register their arrival at the gate | | |
| **Entradas** | **Nombre de entrada** | **Tipo de Dato** | **Condición de selección o repetición** |
| | passengerName | String | the search method must find the passenger |
| | | | |
| **Actividades generales necesarias para obtener los resultados** | The System search for the passenger in the list, if the passenger is in the list, the system will register the arrival to the gate | | |
| **Resultado o Postcondición** | The passenger is given a value based on the arrival order, and a confirmation or error message is printed | | |
| **Salidas** | **Nombre salida** | **Tipo de dato** | **Condición de selección o repetición** |
| | msg | String | |

| | | | |
|---|---|---|---|
| **Nombre o identificador** | **R4: Show boarding order** | | |
| **Resumen** | Once the aircraft doors are opened, the system must allow the crew member in charge to be shown in which order the passengers should board.<br><br>Passengers in first class board first , taking into account the order of arrival, accumulated miles, special attention required, elderly, traveling with baby<br><br>The other passengers enter the plane in the order of arrival but starting with those furthest from the entrance door to the one closest to it. | | |
| **Entradas** | **Nombre de entrada** | **Tipo de Dato** | **Condición de selección o repetición** |
| | - | - | - |
| **Actividades generales necesarias para obtener los resultados** | When the passengers are registered and the user selects the option in the menu to start boarding the plane, the system has to calculate and compare the priorities and other factors to shows the boarding order. | | |
| **Resultado o Postcondición** | The system prints the names of the passengers to show the boarding order | | |
| **Salidas** | **Nombre salida** | **Tipo de dato** | **Condición de selección o repetición** |
| | boardingOrder | String | |

| | | | |
|---|---|---|---|
| **Nombre o identificador** | **R5: Show unboarding order** | | |
| **Resumen** | The flight crew person in charge of the system will be able to see in which order the passengers should exit. Those who leave first are the ones in the first rows and for each row the order is established by proximity to the aisle or order of arrival as the last instance. | | |
| **Entradas** | **Nombre de entrada** | **Tipo de Dato** | **Condición de selección o repetición** |

| | | | |
|---|---|---|---|
| | - | - | |
| **Actividades generales necesarias para obtener los resultados** | The system will calculate the unboarding order, based on the given conditions. | | |
| **Resultado o Postcondición** | The system will show the name of the passengers in order to get out of the plane | | |
| **Salidas** | **Nombre salida** | **Tipo de dato** | **Condición de selección o repetición** |
| | unboardingorder | String | |

**FASE 1: IDENTIFICACIÓN DEL PROBLEMA**

A recognized Airline needs a program to help them in some tasks inside the passengers management, inside those requirements we´ve got to load the passengers info by a TXT file, also the program will be able to search a Passenger´s info by typing it´s name.

The program should be able to register a passenger´s arrival to the gate, and show the boarding and unboarding order according to the priority calculated.

**FASE 2: RECOPILACIÓN DE LA INFORMACIÓN NECESARIA**

1. Algorithm for loading passenger information from a plain text file.
2. Efficient information search algorithm for a passenger.
3. Algorithm for registering the arrival of a passenger at the boarding gate.
4. Passenger ordering algorithm to show the order of entry to the plane.
5. Passenger ordering algorithm to show the order of departure of the plane.
6. Analysis of temporal and spatial complexity of at least two of the implemented algorithms.

Hash Tables: Hash tables in Java are data structures that allow efficient access and search of elements in a collection using a unique key associated with each element. They are based on the use of a hash function that takes the element's key and converts it into an index in the hash table.

```java
package hashtables;

public interface IHashTable<K,V> {
  public void insert(K key, V value) throws Exception;
  public V search(K key);
  public void delete(K key);
}
```

```java
package hashtables;

public class HNode<K,V> {

  private V value;
  private K key;

  private HNode<K,V> next;
  private HNode<K,V> previous;


  public HNode(K key, V value) {
    this.value = value;
    this.key = key;
    next = null;
    previous = null;
  }
```

```java
package hashtables;

import java.util.ArrayList;

public class OpenHashTable<K,V> implements IHashTable<K,V> {

  private ArrayList<HNode<K,V>> table;
  private HNode<K,V> deleted;
```

¿Cómo se especifica un TAD de manera formal?

| **TAD** ⟨*nombre*⟩ |
|---|
| ⟨*objeto abstracto*⟩ |
| { *inv:* ⟨*Invariante del TAD*⟩ } |
| Operaciones Primitivas:<br>▪ ⟨*Operación 1*⟩ : ⟨*Entradas*⟩ → ⟨*Salida*⟩<br>▪ . . .<br>▪ ⟨*Operación n*⟩ : ⟨*Entradas*⟩ → ⟨*Salida*⟩ |

## FASE 3: BÚSQUEDA DE SOLUCIONES CREATIVAS

### lluvia de ideas:

- compare the priority of the passengers to make the boarding list
- separate the plane by sections/rows
- give a priority score where the first class passengers always have 1 more than the normal passengers
- put all the passengers in a single stack already organized ready to unboard
- make different queues for every row of the plane
- make different txt documents for the passenger registry and the passenger and plane info
- make one txt doc with all the data

## FASE 4: TRANSICIÓN DE LA FORMULACIÓN DE IDEAS A LOS DISEÑOS PRELIMINARES

- compare the priority of the passengers to make the boarding list

This idea may have too many unnecesary comparisons because a passenger from the row 10 will never go before the passenger in the row 12 when boarding, and a first class wilil always go before the regular class.

- give a priority score where the first class passengers always have 1 more than the normal passengers

The regular class passengers dont need a priority, they just go based on their row and arrival order.

- put all the passengers in a single stack already organized ready to unboard

separating stacks will be easier and faster

**FASE 5: EVALUACIÓN Y SELECCIÓN DE LA MEJOR SOLUCIÓN**

- make one txt doc with all the data

making one document with all the necesary information divided in well specified sections will be a better solution to avoid reading different documents and an easier way of writting the data

- separate the plane by sections/rows

this will make the process easier and will make the process of makinng the boarding and unboarding order faster

**FASE 6: PREPARACIÓN DE INFORMES Y ESPECIFICACIONES**

**FASE 7: IMPLEMENTACIÓN DEL DISEÑO**

2. Análisis.
    a. Análisis de complejidad temporal de al menos dos de los algoritmos implementados.
    b. Análisis de complejidad espacial de al menos dos de los algoritmos implementados.

**Análisis de complejidad del algoritmo implementado para hacer push a la cola de prioridad (PriorityQ):**

```java
class Node<V> {

    V data;
    double priority;
    int order;
    Node next; // el siguiente con menor prioridad

    public Node(V d, int o) {
        data = d;
        order = o;
        next = null;
    }
    public Node(V d, double p, int o) {
        data = d;
        priority = p;
        order = o;
        next = null;
    }
    public V getData() {
        return data;
    }

}
```

```java
    public void push(V d, int o) {
        Node<V> temp = new Node(d, o);
        if(head == null) {
            head = temp;
        } else {
            Node<V> start = head;
            Node <V> prev = null;

            while (start != null && start.order > o) {
                prev = start;
                start = start.next;
            }
            if(prev == null){
                head = temp;
            } else {
                prev.next = temp;
            }
            temp.next = start;
        }
    }
```

**Temporal:**

The algorithm implements the push method of a priority queue in ascending order using linked lists, assuming that the priority queue has N elements, the analysis of the temporal complexity in the worst case is carried out as follows:

- From lines 3 to 25 (class implemented to represent elements as nodes of the linked list) and 40 to 45, we can see that they are declarations, assignments, or comparisons, therefore the complexity of these fragments is O(1).
- From lines 47 to 49, the search for the element with priority lower than or equal to the node to be added is performed, in the worst case all elements in the queue have higher priority than the node to be added, therefore, it will have to traverse the N elements in the linked list to conclude that it goes to the end of the list and add it. It can be deduced that the complexity is O(N).
- From lines 51 to 56, we can see that they are assignments or comparisons, therefore the complexity of these fragments is O(1).

It can be deduced that the temporal complexity in the worst case of the implemented algorithm to push elements into a priority queue in ascending order using linked lists is O(N).

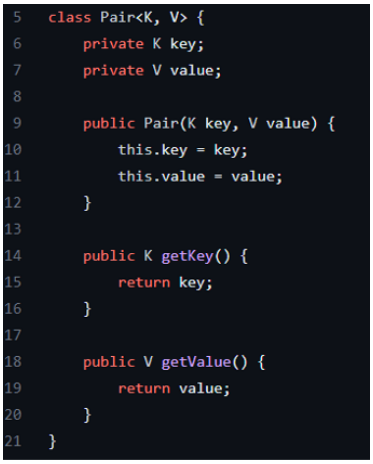
**Spatial:**

As mentioned in the temporal analysis, a linked list is used to store the nodes of the priority queue, then the analysis of the spatial complexity in the worst case is as follows:

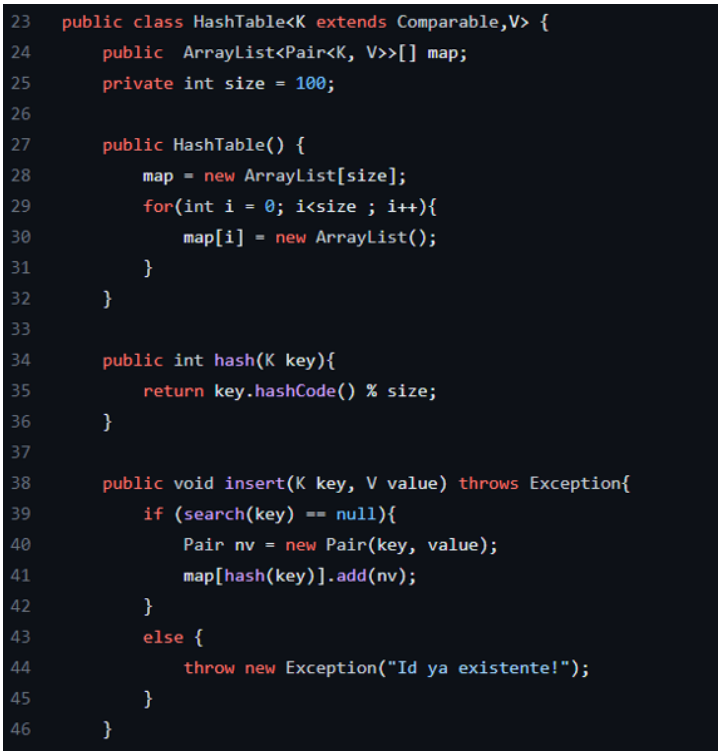
- Building each node requires a constant amount of parameters (lines 3 to 25), therefore the spatial complexity is O(1).
- As each node is stored independently in the linked list, N elements in the linked list are required to store N nodes, therefore the temporal complexity is O(N).

It can be deduced that the spatial complexity in the worst case of the implemented algorithm to push elements into a priority queue in ascending order using linked lists is O(N).

## Análisis de complejidad del algoritmo implementado para hacer insert al hastable (HashTable):



```
5   class Pair<K, V> {
6       private K key;
7       private V value;
8
9       public Pair(K key, V value) {
10          this.key = key;
11          this.value = value;
12      }
13
14      public K getKey() {
15          return key;
16      }
17
18      public V getValue() {
19          return value;
20      }
21  }
```



```
23  public class HashTable<K extends Comparable,V> {
24      public  ArrayList<Pair<K, V>>[] map;
25      private int size = 100;
26
27      public HashTable() {
28          map = new ArrayList[size];
29          for(int i = 0; i<size ; i++){
30              map[i] = new ArrayList();
31          }
32      }
33
34      public int hash(K key){
35          return key.hashCode() % size;
36      }
37
38      public void insert(K key, V value) throws Exception{
39          if (search(key) == null){
40              Pair nv = new Pair(key, value);
41              map[hash(key)].add(nv);
42          }
43          else {
44              throw new Exception("Id ya existente!");
45          }
46      }
```

**Temporal:**

The shown algorithm implements the insert method of a hash table using ArrayLists, assuming the hash table has N elements, the analysis of the temporal complexity in the worst case is as follows:

- From lines 5 to 21 (class implemented to represent elements as pairs of elements) and 39 to 40, we can see that they are declarations, assignments, or comparisons, therefore the complexity of these fragments is O(1).
- In line 41, the hash function (implemented in lines 34 to 36) is used to calculate the index in the ArrayList array of the element to be added, this operation has temporal complexity O(1) since only the hashCode method call and the modulo operator are used. On the other hand, to insert the element to the found ArrayList, the add function is used, which has an amortized complexity of O(1), given that in most of the executions adding an element using the add function that receives only one parameter will have complexity O(1), but rarely will have complexity O(N) because in the implementation used, operations of reallocation of all elements will have to be made to make space for the new one. Therefore, the complexity of line 41 is O(1) amortized.
- From lines 42 to 46, we can see that they are assignments or comparisons, therefore the complexity of these fragments is O(1).

It is deduced that the temporal complexity in the worst case of the implemented algorithm to insert into the hash table is O(1) amortized.

**Espacial:**

As mentioned in the temporal analysis, an array (of constant length) is used to store the ArrayLists containing key-value pairs of the hash table, so the analysis of spatial complexity in the worst case is as follows:

- Constructing each element of the ArrayList (the key-value pairs of the Pair class) requires a constant number of parameters (lines 5 to 21), so the spatial complexity is O(1).
- Since each ArrayList, in the worst case, contains all the elements added to the hash table, which would be the case if they all have the same hash, the spatial complexity would be O(N).
- Since the array of ArrayLists is a structure with a fixed number of elements, the spatial complexity would be O(1).

It follows that the spatial complexity in the worst case of the implemented algorithm for inserting into the hash table using ArrayLists is O(N).

**3.** Diseño.

**a.** Diseño del TAD de las estructuras de datos utilizadas.

| **TAD HashTable** |
| --- |
| HashTable : { ArrayList<Pair<K, V>>} |
| { inv: size } |
| Primitive Operations:<br>▪ HashTable: size → Hashmap<br>▪ hash: key → hashvalue<br>▪ insert: TablaHash x key x value → Void<br>▪ search: TablaHash x Key → value |

| **HashTable()**<br>"Creates the hashmap"<br>{ pre: map → null}<br>{ post: map → true } |
| --- |

| **hash()**<br>"converts a key into a hashcode" |
| --- |

{ pre: key }
{ post: hashcode }

**insert()**
"adds a passenger to the hashmap"
{ pre: map → true ∧ passenger ∉ map}
{ post: map → true ∧ passenger ∈ map}

**search()**
"searches for a passenger in the hashmap given a key"
{ pre: map → true ∧ key }
{ post: passenger V null }

**TAD PriorityQ**

PriorityQ : { N1<V> , N2<V>... }

{ inv: head.priority ≥ head.next.priority }

Primitive Operations:
- push: Value x Order → Void
- push: Value x Priority xOrder → Void
- isEmpty: head → boolean
- pop: head → Value

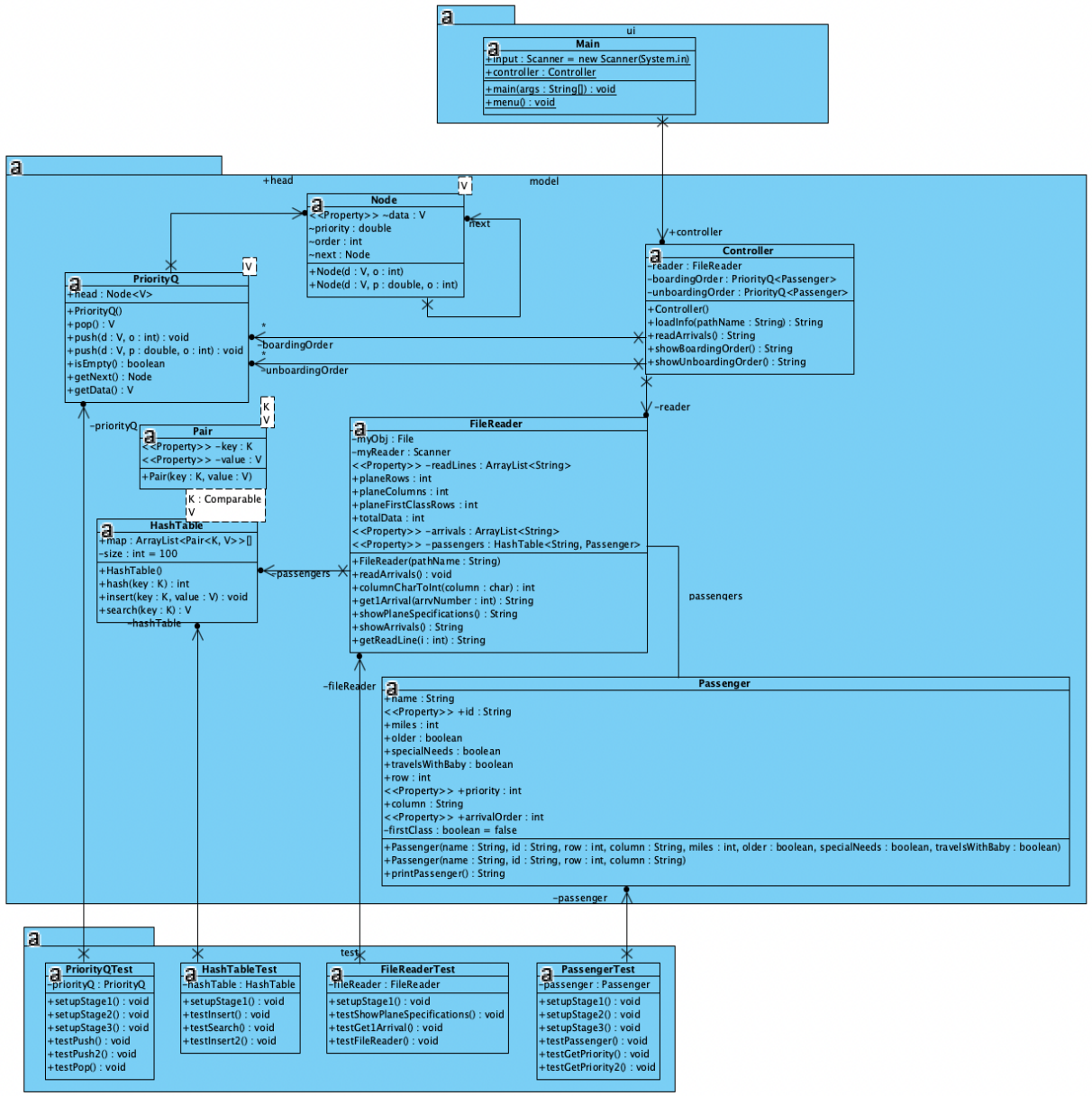
**push()**
"Adds a passenger to the queue"
{ pre: PriorityQ → true ∧ passenger ∉ PriorityQ}
{ post: PriorityQ → passenger ∧ passenger ∈ PriorityQ }

**isEmpty()**
"Checks if the queue is empty"
{ pre: PriorityQ→ true }
{ post: true or false }

**pop()**

"removes the top passenger from the queue"
{ pre: PriorityQ → passenger}
{ post: passenger}

**b.** Diseño completo del diagrama de clases usando generics, incluyendo, las estructuras de datos, el paquete del modelo, de la interfaz de usuario y pruebas.



**c.** Diseño de casos de prueba, incluyendo los escenarios, para las estructuras y el sistema.

## Formato de escenarios y casos de uso

**Configuración de los Escenarios**

| Nombre | Clase | Escenario |
|---|---|---|
| setupStage 1 | *Passenger***Test** | A non initialized object Passenger |
| setupStage 2 | *Passenger***Test** | An object of the class Passenger is created with name=”Lucas”, id=”1193”, row=2, column=”A”, miles=200, older=false, specialNeeds=true, travelsWithBaby=false |
| setupStage 3 | *Passenger***Test** | An object of the class Passenger with name=”Julian”, id=”2143”, row=5, column=”A” |
| setupStage 1 | *FIleReaderTest* | An object of the class FileReader with pathName=”Data.txt” |
| setupStage 1 | *HashTableTest* | An object of the class HashTable with 3 pairs of K,V:<br>where V is an object of the class Passenger<br>1. key= 1193, y value con name=”Lucas”, id=”1193”, row=2, column=”A”, miles=200, older=false, specialNeeds=true, travelsWithBaby=false<br>2. key= 2143, value con name=”Julian”, id=”2143”, row=5, column=”A”<br>3. key=5493, value con con name=”Rodrigo”, id=”5493”, row=16, column=”C” |
| setupStage 1 | *PriorityQTest* | An object of the class PriorityQ |
| setupStage 2 | *PriorityQTest* | An object of the class PriorityQ with 1 node:<br>1. d=-3<br>p= an object passenger with:<br>name=”Lucas”, id=”1193”, row=2, column=”A”, miles=200, older=false, specialNeeds=true, travelsWithBaby=false |
| setupStage 3 | *PriorityQTest* | An object of the class PriorityQ with 2 nodes:<br>2. d=-3<br>p= an object passenger with:<br>name=”Lucas”, id=”1193”, row=2, column=”A”, miles=200, older=false, specialNeeds=true, travelsWithBaby=false<br>**3. d=-5**<br>p= an object passenger with:<br>name=”Julian”, id=”2143”, row=2, column=”B”, **miles=300**, older=false, specialNeeds=true, travelsWithBaby=false |

**Diseño de Casos de Prueba**

| **Objetivo de la Prueba:** Verify that the methods from the class Pasenger are effective | | | | |
|---|---|---|---|---|
| **Clase** | **Método** | **Escenario** | **Valores de Entrada** | **Resultado esperado** |
| Passenger | Passenger | setupStage1 | name=”Lucas”, id=”1193”, row=2, column=”A”, miles=200, older=false, specialNeeds=true, travelsWithBaby=false | A new Passenger created with the given parameters. |
| Passenger | getPriority | setupStage2 | — | returns the priority of the passenger priority=3 |
| Passenger | getPriority | setupStage3 | — | returns priority=0 because the passenger is not first class so it doesn't have priority |

| **Objetivo de la Prueba:** Verify that the methods in FileReader class work correctly | | | | |
|---|---|---|---|---|
| **Clase** | **Método** | **Escenario** | **Valores de Entrada** | **Resultado esperado** |
| FileReader | showPlaneSpecifications | setupStage1 | the first 4 lines of the Data.txt document must be:<br><br>/**** Plane definition section ****/<br>4 // Rows in the plane:<br>D // Columns on the plane<br>2 // First Class<br><br>which indicates the specifications of the plane | A String=”Plane rows: 4  Plane columns: 4  Plane first class: 2” |
| FileReader | get1Arrival | setupStage1 | **arrvNumber=1**<br><br>The arrival section in the Data.txt document must be:<br><br>/**** Arrival ****/<br>010<br>016<br>011 | returns the id of the passenger to arrive first= 010 |
| FileReader | getReadLine | setupStage1 | i=6 | Returns the line 6 in the Data.txt file “/**** model.Passenger Database ****/“ |

| **Objetivo de la Prueba:** Verify that the insert and search  methods from the class HashTable work fine | | | | |
|---|---|---|---|---|
| **Clase** | **Método** | **Escenario** | **Valores de Entrada** | **Resultado esperado** |

| | | | | |
|---|---|---|---|---|
| HashTable | insert | setupStage 1 | key=”1193” value=Passenger obj with: name=”Jimmy”, id=”1193”, row=4, column=”B”, miles=100, older=true, specialNeeds=true, travelsWithBaby=false | throws an Exception with the message "Id ya existente!" |
| HashTable | search | setupStage 1 | key=1193 | Returns the object Passenger with the id “1193” |
| HashTable | insert | setupStage 1 | key=”2340” value=Passenger obj with: name=”Jimmy”, id=”2340”, row=4, column=”B”, miles=100, older=true, specialNeeds=true, travelsWithBaby=false | inserts a pair of a key and a passenger in the hashmap |

| **Objetivo de la Prueba:** Verify that the methods from the class PriorityQ work fine | | | | |
|---|---|---|---|---|
| **Clase** | **Método** | **Escenario** | **Valores de Entrada** | **Resultado esperado** |
| PriorityQ | push | setupStage 1 | d=1<br>p= an object passenger with:<br>name=”Julian”, id=”2143”, row=2, column=”B”, miles=200, older=false, specialNeeds=true, travelsWithBaby=false | the priorityQ has 1 node, the head is not null. The next of the head is null<br>head data is an object passenger with:<br>name=”Julian”, id=”2143”, row=2, column=”B”, miles=200, older=false, specialNeeds=true, travelsWithBaby=false |
| PriorityQ | push | setupStage 2 | d=1<br>p= an object passenger with:<br>name=”Julian”, id=”2143”, row=2, column=”B”, **miles=200**, older=false, specialNeeds=true, travelsWithBaby=false | the priorityQ has 2 nodes, the new head is the object passenger pushed:<br>name=”Julian”, id=”2143”, row=2, column=”B”, **miles=200**, older=false, specialNeeds=true, travelsWithBaby=false |
| PriorityQ | pop | setupStage 3 | — | returns the head of the queue which is an obj Passenger:<br>name=”Julian”, id=”2143”, row=2, column=”B”, **miles=300,** older=false, specialNeeds=true, travelsWithBaby=false |

4. Implementación en Java.
    a. Implementación completa de las estructuras de datos y sus pruebas.
    b. Implementación completa y correcta del modelo, la UI y las pruebas (contexto del problema).

link de github:
https://github.com/felipevelasco7/APO-2/tree/master/TI1_CyED