Integrative task I
Computing and Discrete Structures I
Development of the Engineering Method

Members:

- Juan Jose Diaz A00381098
- Ana Sofía Londoño A00380882

1. Implementing the engineering method:

Step 1 - Identification of the problem.

Context of the problem
A recognized Health Provider Institution requires the creation of a system that allows managing the entry and exit of patients in a Clinical Laboratory. This program is expected to be executed by the personnel located at the reception of said center who will be in charge of carrying out the admission process and later directing the person to one of the two current units of the laboratory (Hematology and General Purpose) with a shift of assigned care.

Problem Definition
A service provider institution has requested a program that allows managing the admission and discharge of patients in a clinical laboratory.

Definition of needs
- The hospital needs to store the information of its patients in a database, which they can access at any time
  - Must be able to add, search and undo information about patients in this database
- The solution must allow patient care in a differentiated way, depending on whether it is a priority or non-priority patient
- The software must allow the management of the list of people who are currently in the laboratory.

Step 2 - Information gathering:

To address the problem of patient management in this clinical laboratory from software engineering, we are first going to raise some important concepts that will be relevant when implementing the software.

We have to find data structures which allow us to store the potentially large amount of patient information with a relatively fast access capacity and also structures that can simulate the management of this information efficiently, such as adding, undoing, and manage patients who go to the laboratory in real time

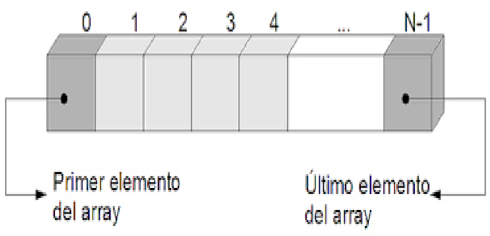
Definitions:

Stacks: Ordered list or database that is responsible for storing or retrieving data through the LIFO access mode (Last In - First Out). (Sumo Logic, Inc., 2021)
Queue: Abstract data type that is characterized by being a sequence of elements that are ordered by means of the FIFO access method. (Yang, 2021)
Priority queues: Priority Queue is a data structure more specialized than Queue. Unlike ordinary queues, Priority queue's items are ordered by key value so that item with the lowest value of key is at front and item with the highest value of key is at rear end or vice versa. So we assigned priority to the item based on its key value. (GeeksforGeeks, 2022)
Hash tables: Is a data structure used to implement an associative array, a structure that can map keys to values. A hash table uses a hash function to compute an index into an array of buckets or slots, from which the desired value can be found. (Stemmler, 2022)

Step 3 -Search for creative solutions:

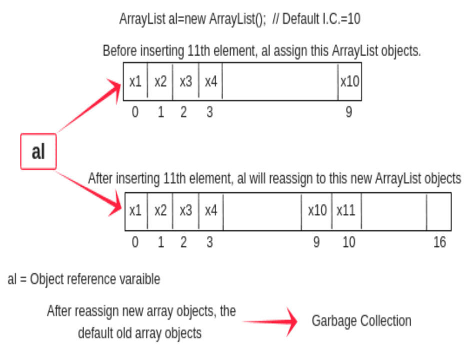
1. We identify that for the storage of patient information, we have 3 alternatives which we can implement in our solution:

Alternative 1. Fixed size arrays:



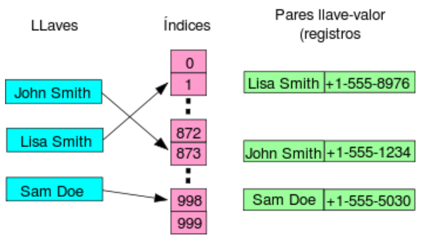
A data structure that allows us to store a limited amount of data of the same type. This data structure has the condition of being initialized with a fixed size. To be able to access the element inside an array you need to have an index. This can be achieved by means of some repetitive instance. (Braunschweig, 2018)

Alternative 2. Dynamic arrays:


ArrayList al=new ArrayList(); // Default I.C.=10
Before inserting 11th element, al assign this ArrayList objects.
x1 x2 x3 x4 x10
0 1 2 3 9
al
After inserting 11th element, al will reassign to this new ArrayList objects
x1 x2 x3 x4 x10 x11
0 1 2 3 9 10 16
al = Object reference varaible
After reassign new array objects, the default old array objects
Garbage Collection

Dynamic array or arraylist is a data storage structure which is characterized by increasing or decreasing its capacity or size according to the elements that are added.This data structure has its own methods, however to add, find or delete an element also requires repetitive instances. (Dynamic Array in Java - Javatpoint, s. f.)

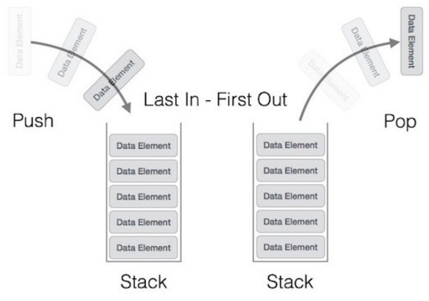
Alternative 3. Hash Tables.



A data structure that allows us to associate keys with values of any type. The braces are considered as indices or positions to store, save or remove those values within a specific position. In order for the keys to become indexes, it is necessary to pass them through functions specialized in distributing the space to which they are going to be assigned. (Stemmler, 2022)

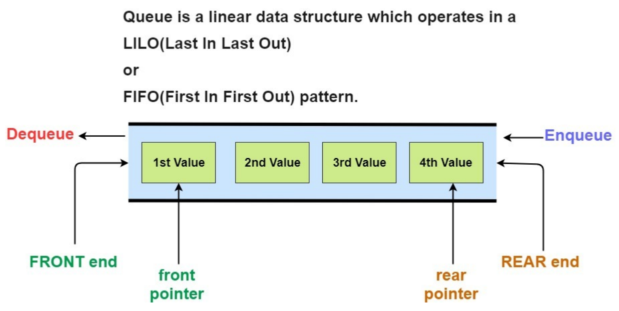
2. For the care of patients in the hospital, we need a data structure that allows us to manage patients in a differentiated way according to the priority that each one has, such as the elderly, pregnant women, serious illness, etc... For this we identify that we have the following 3 options:

Alternative 1. Stacks:



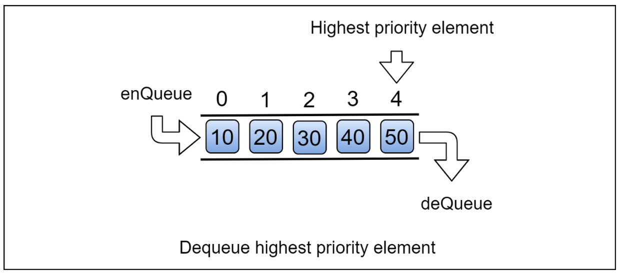
Stacks in Data Structures is a linear type of data structure that follows the LIFO (Last-In-First-Out) principle and allows insertion and deletion operations from one end of the stack data structure, that is top. (Semilof & Montgomery, 2020)

Alternative 2. Queue:


Queue is a linear data structure which operates in a LILO(Last In Last Out) or FIFO(First In First Out) pattern.
Dequeue
Enqueue
1st Value
2nd Value
3rd Value
4th Value
FRONT end
front pointer
rear pointer
REAR end

A queue is a data structure in which whatever comes first will go out first, and it follows the FIFO (First-In-First-Out) policy. It's defined as the list or collection in which the insertion is done from one end known as the rear end or the tail of the queue, whereas the deletion is done from another end known as the front end or the head of the queue. (Yang, 2021)

Alternative 3. Priority Queue:



A priority queue is a special type of queue in which each element is associated with a priority value. And, elements are served on the basis of their priority. That is, higher priority elements are served first. (GeeksforGeeks, 2022)

Step 4 - Transition from Ideas to Preliminary Designs

The first thing we do in this step is to discard the ideas that are not feasible. First, for the storage of patient information, we have to take into account two things, the first is that we do not have an exact number of all the possible patients that are going to go to the hospital, that is why we discard the fixed sized arrangement. The second thing to keep in mind is the complexity of accessing the information. Although dynamic arrays are a good option when storing an unknown number of data, the information access time is much higher compared to other alternatives, which is the reason why we discard dynamic arrays.

Secondly, to attend the patients we need a data structure that allows us to manage the patients in a differentiated way according to the priority that each one has, that is why we discard the stack, and the queue data structure since they don't allow us to achieve the objective of differently managing a queue of people with different priorities.

Step 5 - Evaluation and Selection of the Best Solution

Criteria.

Criterion A: Adding patient efficiency (Alternative Efficiency).
- **[1] Lineal O(n)**
- **[2]Logarithmic O(n Log(n))**
- **[3]Logarithmic O(Log(n))**
- **[4] Constant O(1)**

Criterion B: Accessing to patient efficiency (Alternative Efficiency).
- **[1] Lineal O(n)**
- **[2]Logarithmic O(n Log(n))**
- **[3]Logarithmic O(Log(n))**
- **[4] Constant O(1)**

Criterion C: Delete patient efficiency (Alternative Efficiency).
- **[1] Lineal O(n)**
- **[2]Logarithmic O(n Log(n))**
- **[3]Logarithmic O(Log(n))**
- **[4] Constant O(1)**

	Criterion A	**Criterion B**	**Criterion C**	**Total**

Alternative 1. Dynamic arrays	**Lineal O(n)** **1**	**Lineal O(n)** **1**	**Lineal O(n)** **1**	**3**
Alternative 2. Hash Tables	**Constant O(1)** **4**	**Constant O(1)** **4**	**Constant O(1)** **4**	**12**

To register patients we use hash tables, since this is a data structure that may need low complexity when accessing the information of each patient.
Criteria.

Criterion A: Adding patient efficiency (Alternative Efficiency).
- **[1] Lineal O(n)**
- **[2] Logarithmic O(n Log(n))**
- **[3] Logarithmic O(Log(n))**
- **[4] Constant O(1)**

Criterion B: Accessing to patient efficiency (Alternative Efficiency).
- **[1] Lineal O(n)**
- **[2] Logarithmic O(n Log(n))**
- **[3] Logarithmic O(Log(n))**
- **[4] Constant O(1)**

Criterion C: Delete patient efficiency (Alternative Efficiency).
- **[1] Lineal O(n)**
- **[2] Logarithmic O(n Log(n))**
- **[3] Logarithmic O(Log(n))**
- **[4] Constant O(1)**

Criterion D: Allows treatment of patients with different priorities.
- **[0] No**
- **[1] Yes**

	Criterion A	**Criterion B**	**Criterion C**	**Criterion D**	**Total**
Alternative 2. Queue	**Logarithmic O(Log(n))** **3**	**Constant O(1)** **4**	**Logarithmic O(Log(n))** **3**	**No** **0**	**10**
Alternative 3.	**Logarithmic O(Log(n))**	**Constant O(1)**	**Logarithmic O(Log(n))**	**Yes** **1**	**11**

Priority Queues	**3**	**4**	**3**		

To manage patients we use priority queues, since this is a data structure that allows us to achieve the objective of managing a queue of people with different priorities.

Selection:
With respect to the previous evaluation, Alternative 2 was selected, which corresponds to Hash Tables because it has the highest score and has the highest efficiency.

Step 6 - Preparation of reports and specifications:

Since we have a better understanding of the structures that we are going to use and the scope that our solution can take from these, we will be able to better define the requirements of the problem of managing patients in the clinical laboratory.

Functional requirements:

R1:Database: The system has to have a database which stores the information of all the users. This database has to be simulated by an initial data load of the patients through a plain text file.

R2:Search or entrance of a patient: The system has to be able to search or register a patient in case it isn't registered yet, it must be the most efficient way possible.

R3:Type of attention: The type of attention given to the patients has to be in two ways. Priority type: For those who have an important underlying disease, older adults, pregnant women, etc. General access type: For those who don't have any kind of priority and it will work by the rule of first come first serve.

R4: Undo option: The system must have an option to “undo” option available each time an input or output action is performed, to enable the correction of these possible mistakes.

R5: Display panel: The system must have a panel that monitors at all times the list of people currently in the laboratory, the order of care of the people in each unit and, of course, the action of patient discharge to continue with the care of other people waiting to perform their respective tests.

R6: (Bonus) Automatic exit within the units: The exit within the units must be automatic after a random time of between 1 or 2 minutes to simulate the real attention of the laboratory. After which the person will not appear waiting to be served in the unit, but must still be present in the system to manually exit the location by a reception person.

R7: (Bonus) Graphical User Interface (GUI): The system must have a Graphical User Interface that allows the interaction between the system and the user to be more dynamic

Analysis

Temporal complexity Analysis

show content in a Hash Table algorithm:

Statement	*Effort*
ArrayList<V> nodesInTheTable = new ArrayList<>();	*1*
for(int i = 0; i< table.length; i++)	*n+1*
if(table[i] != null)	*n*
HashNode<K, V> temporalNode = table[i];	*n*
while (temporalNode != null)	*n (n+1)*
nodesInTheTable.add(temporalNode.getValue());	*n * n*
temporalNode = temporalNode.getNext();	*n * n*
return nodesInTheTable;	*1*

$$T(A) = 1 + n + 1 + n + n(n + 1) + n^2 + n^2 + 1$$
$$T(A) = 3 + 3n + 3n^2$$

With this we can say that the time complexity of this algorithm in big O notation would be : $O(n^2)$

Save data algorithm

Statement	*Effort*
String out = patient.getId() + "," + patient.getName() + "," + patient.getGender() + "," +patient.getAge() + "," +patient.isPrioritized() + "," + patient.getPriorityValue() + "\n";	***1***
File file = new File("DataBase.csv");	***1***
FileInputStream fis = new FileInputStream(file);	***1***
BufferedReader reader = new BufferedReader(new	***1***

InputStreamReader(fis));	
String line;	***1***
while ((line = reader.readLine()) != null) {	***n+1***
out += line + "\n";	***n***
FileOutputStream fos = new FileOutputStream(file);	***n***
fos.write(out.getBytes(StandardCharsets.UTF_8));	***n***
fos.close();	***n***

$$T(A) = n + 1 + n + 13$$
$$T(A) = 2n + 14$$

With this we can say that the time complexity of this algorithm in big O notation would be : $O(n)$

Spatial complexity Analysis

Statement

public void addPatient(String id, String name, int g, int age, boolean isPrioritized, int priorityValue) {
    Patient patient = new Patient(id,name,assingGender(g),age,isPrioritized, priorityValue);
    dataBase.insert(id,patient);
    saveData(patient);
}

Type	***Variable***	***Lenght***	***Amount Values***
Input	*id*	-	*0*
	name	-	*0*
	gender	*32*	*1*
	age	*32*	*1*
	isPrioritized	*16*	*1*
	priorityValue	*32*	*1*
Aux	*patient*	-	*0*
Output	*none*	-	

$$input\ +Aux+Output\ =\ 4=O(1)$$

The spatial complexity of this algorithm is $O(1)$

Statement

```
public void addPatientToDataBase() {
        System.out.println("Enter the id of the patient: ");
        String id = sc.nextLine();
        if(clinic.isPatientInDataBase(id)) {
            System.out.println("The patient is already in the dataBase :)");
        } else {
            System.out.println("Enter the name of the patient: ");
            String name = sc.nextLine();
            System.out.println("Enter the gender of the patient:\n1. Male\n2. Female\n3.
Non-Binary");
            int gender = Integer.parseInt(sc.nextLine());
            System.out.println("Enter the age of the patient:");
            int age = Integer.parseInt(sc.nextLine());
            System.out.println("The patient is prioritized?\n1. YES\n2. NO");
            int prioritized = Integer.parseInt(sc.nextLine());
            int priorityValue;
            if(prioritized == 1) {
                System.out.println("From 1 to 5 how urgent is your atention");
                priorityValue = Integer.parseInt(sc.nextLine());
                while(priorityValue<1 || priorityValue>5){
                    System.out.println("Enter a valid option:\nFrom 1 to 5 how urgent is your attention");
                    priorityValue = Integer.parseInt(sc.nextLine());
                }
                clinic.addPatient(id, name, gender, age, true, priorityValue);
            } else {
                priorityValue = 0;
                clinic.addPatient(id, name, gender, age, false, priorityValue);
            }
        }
    }
```

Type	***Variable***	***Lenght***	***Amount Values***
Input	*none*	-	-
Aux	*id*	-	*0*
	name	-	*0*
	gender	*32*	*1*

	age	*32*	*1*
	prioritized	*32*	*1*
	priorityValue	*32*	*1*
Output	*none*	*-*	*0*

$$input\ +\ Aux\ +\ Output\ =\ 4\ =\ O(1)$$

The spatial complexity of this algorithm is $O(1)$

Abstract data types:

PriorityQueue:

ADT of PriorityQueue
PriorityQueue= { Element _1 = <priorityValue, element>, Element_2 = <priorityValue, element> ... Element_n = <priorityValue, element> }
Indeterminacy: For two elements in the queue, x and y, if x has a lower priority value than y, y will be deleted before x
Primitive Operations: ***(Constructor)*** - CreatePriorityQueue: → PriorityQueue ***(Modifiers)*** - Insert PriorityQueue x Element x PriorityValue → PriorityQueue - ExtractMax PriorityQueue → Element - IncreaseKey PriorityQueue x Element x PriorityValue → PriorityQueue ***(Analyzers)*** - ShowElements PriorityQueue → String - Maximum PriorityQueue → Element

Constructor
CreatePriorityQueue() "Create an empty priority queue"

{Pre: null**}**
"The priority queue is null"

{Pos: PriorityQueue= {∅} **}**
"An empty priority queue has been initialized"

Modifiers

Insert(Element_n)
 	
"Add a new item to the priority queue**"**
 	
{Pre: PriorityQueue!= null **}**
"The priority queue must already be initialized"
 	
{Pos: Element _n ∈ PriorityQueue, PriorityQueue **= {** Element _n = <PriorityValue, element>**}**
"Item n has been added to the priority queue"

ExtractMax()
 	
"Fetch the element that has the highest priority value in the priority queue**"**
 	
{Pre: PriorityQueue!= null & PriorityQueue!= {∅} **}**
"The priority queue must already be initialized and it must not be empty"
 	
{Pos: ((Element _i & Element _j**)** ∈ PriorityQueue) & (Element _i > Element _j)→ Element _i ∉ PriorityQueue & Element _j ∈ PriorityQueue,**}**
"The element with the highest priority value was extracted"

IncreaseKey(Element_n, NewPriorityValue)
 	
"Increases the priority value of an item in the priority queue"
 	
{Pre: PriorityQueue!= null & PriorityQueue!= {∅} & element_n ∈ PriorityQueue & NewPriorityValue > CurrentPriorityValue}
"Increases the priority value of an item in the priority queue"
 	
{Pos: Element_n = <NewPriorityValue,element>}
"Updates the priority value of the item in the priority queue"

Analyzers

ShowElements()
 	
"Shows the elements that are in the priority queue**"**
 	
{Pre: PriorityQueue!= null & PriorityQueue!= {∅} **}**

"The priority queue must already be initialized and it must not be empty"

{Pos: PriorityQueue = Element _1,...,Element _n ∈ String**}**
"Shows a string with the information of the elements of the priority queue"

ShowMax()

"Shows the element that has the highest priority value in the priority queue**"**

{Pre: PriorityQueue!= null & PriorityQueue!= {∅} **}**
"The priority queue must already be initialized and it must not be empty"

{Pos: ((Element _i & Element _j**)** ∈ PriorityQueue) & (Element _i > Element _j)**}**
"Shows the element with the highest priority value"

Stack:

ADT of Stack

Stack = { Element _1, Element_2, …, Element_n-1,Element_n**},** n=top

Indeterminacy: For each last element added to the stack, it will be the first element to be popped. Last-In First-Out (LIFO)

Primitive Operations:

(Constructor)

- CreateStack:		→	Stack

(Modifiers)

- Push	Stack x Element	→	Stack
- Pop	Stack	→	Element

(Analyzers)

- Top	Stack	→	Element
- IsEmpty	Stack	→	Boolean

Constructor

CreatePriorityQueue()

"Create an empty stack**"**

{Pre: null**}**
"The stack is null"

{Pos: Stack = {∅} **}**
“An empty stack has been initialized”

Modifiers

Push(Element_n)

“Add a new item to the stack**”**

{Pre: Stack != null **}**
“The stack must already be initialized”

{Pos: Element _n ∈ Stack ^ top=Element _n ^ Stack **= {** Element _1, Element_2, …, Element_n**}**
“Item n has been added to the stack and this has become the new top of the stack”

Pop()

“Fetch the top of the stack”

{Pre: Stack != null & Stack != {∅} **}**
“The stack must already be initialized and it must not be empty”

{Pos: top=Element_n-1 ^ Element_n ∉ Stack ^ Stack **= {** Element _1, Element_2, …, Element_n-1**} return** top}
“The element with at the top of the stack has been extracted”

Analyzers

Top()

“Shows the top of the stack”

{Pre: Stack != null & Stack != {∅} **}**
“The stack must already be initialized and it must not be empty”

{Pos: Stack **= {** Element _1, Element_2, …, Element_n} ^ top=Element_n ^ **return** top}
“Shows the element with at the top of the stack”

IsEmpty()

“Shows the top of the stack”

{Pre: Stack != null **}**
“The stack must already be initialized “

{Pos: return TRUE if the stack is empty, FALSE if the stack is not empty**}**

Hash Table:

ADT of Hash Table

HashTable = { Element _1 = <key, element>, Element_2 = <key, element> ... Element_n = <key, element> **}**

Indeterminacy: There shouldn't be 2 elements with the same key stored in the hash table.

Primitive Operations:

(Constructor)

- CreateHashTable :		→	HashTable

(Modifiers)

- InsertKey	HashTable x Key x Element	→	HashTable
- DeleteKey	HashTable x Key	→	HashTable

(Analyzers)

- ShowElements	HashTable	→	String
- Search	HashTable x Key	→	Element

Constructor

CreateHashTable(size)

{Pre: null**}**
"The hash table is null"

{Pos: hashTable= {∅} ^ size(Hashtable)=size **}**
"An empty hashTable has been initialized with the size entered"

Modifiers

InsertKey(Key,Element)

"Add a new item to the hash table**"**

{Pre: hashTable!= null **}**
"The hash table must already be initialized"

{HashTable = { Element _1 = <"","">, Element_2 = <"", ""> ... Element_size-1 = <key, element> **}}**
"A key-value pair has been stored in the hash table"

DeleteKey(Key)

“Delete an element from the hash table**”**

{Pre: hashTable!= null ^ hashTable!= {∅} key ∈ hashTable}
“The hash table must already be initialized, must not be empty and the key must exist in the hash table”

{HashTable = { Element _1 = <””,””>, Element_2 = <””, “”> ... Element_size-1 = <””, “”> }}
“A key-value pair has been removed from the hash table”

Analyzers

ShowElements()

“Shows the elements that are in the hash table**”**

{Pre: hashTable!= null & hashTable!= {∅} }
“The hash table must already be initialized and it must not be empty”

{Pos: HashTable= Element _1,...,Element _n ∈ String ^ **return** String}
“Shows a string with the information of the elements of the hash table”

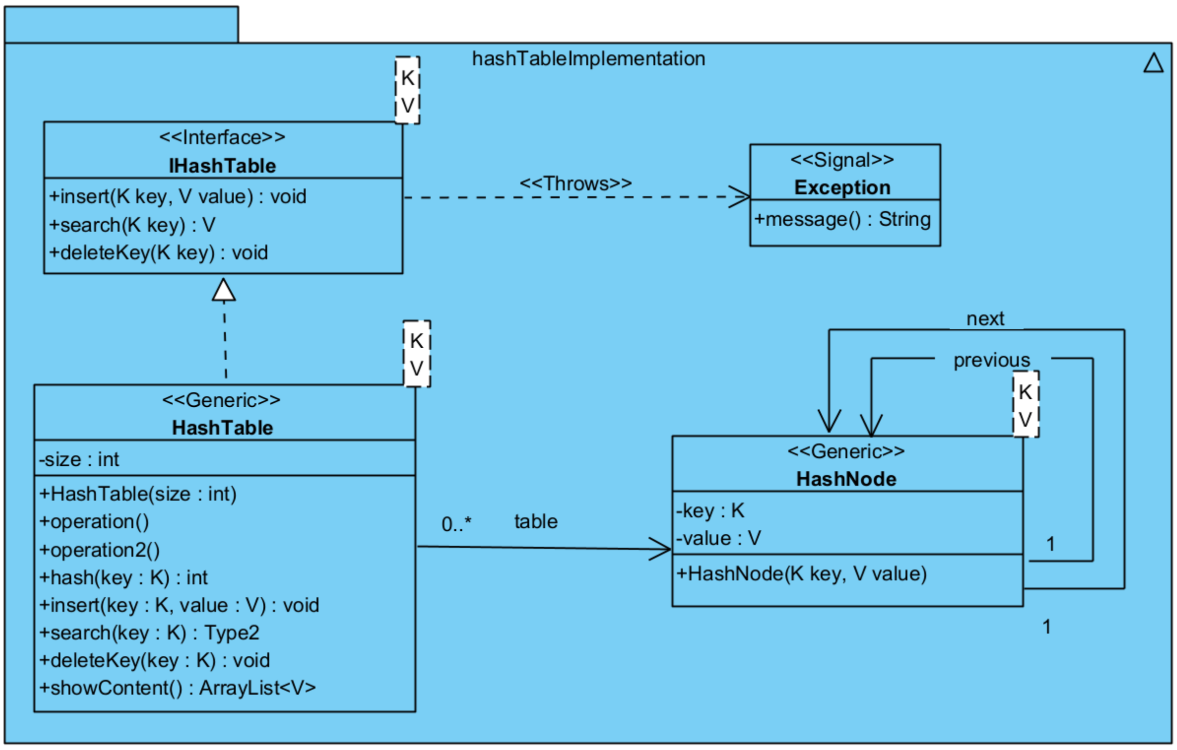
SearchKey(Key)

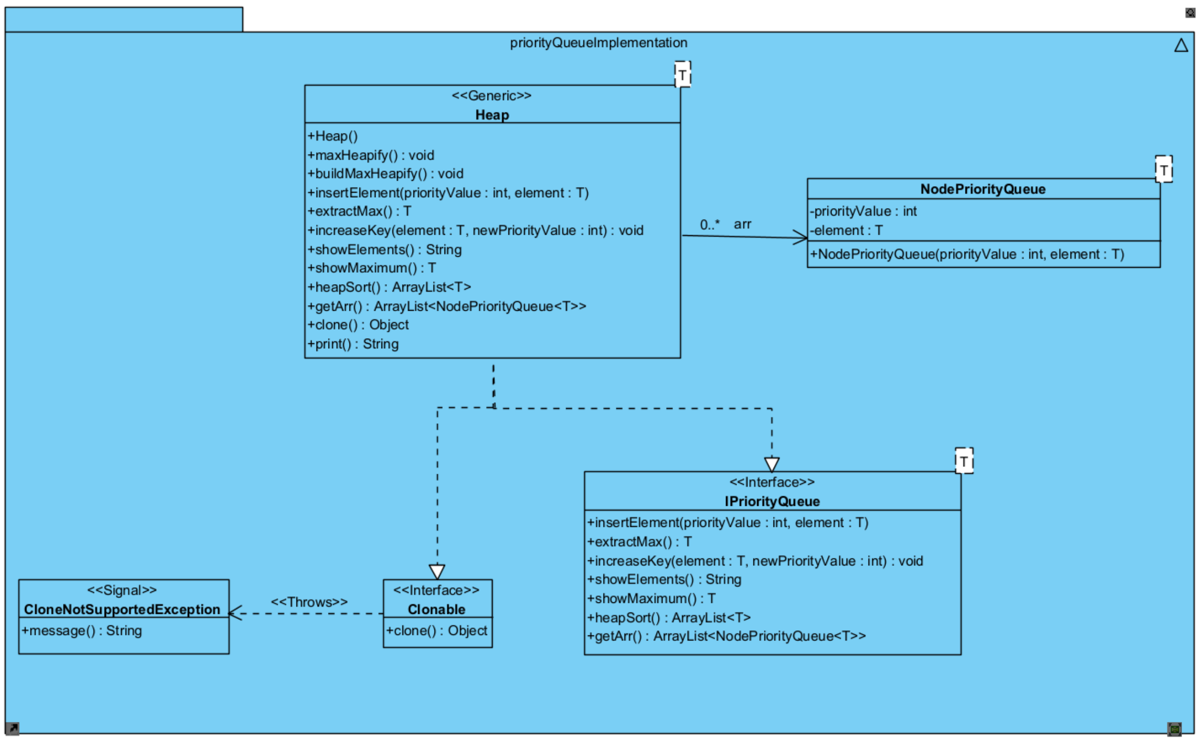
“Search a key from the hash table**”**

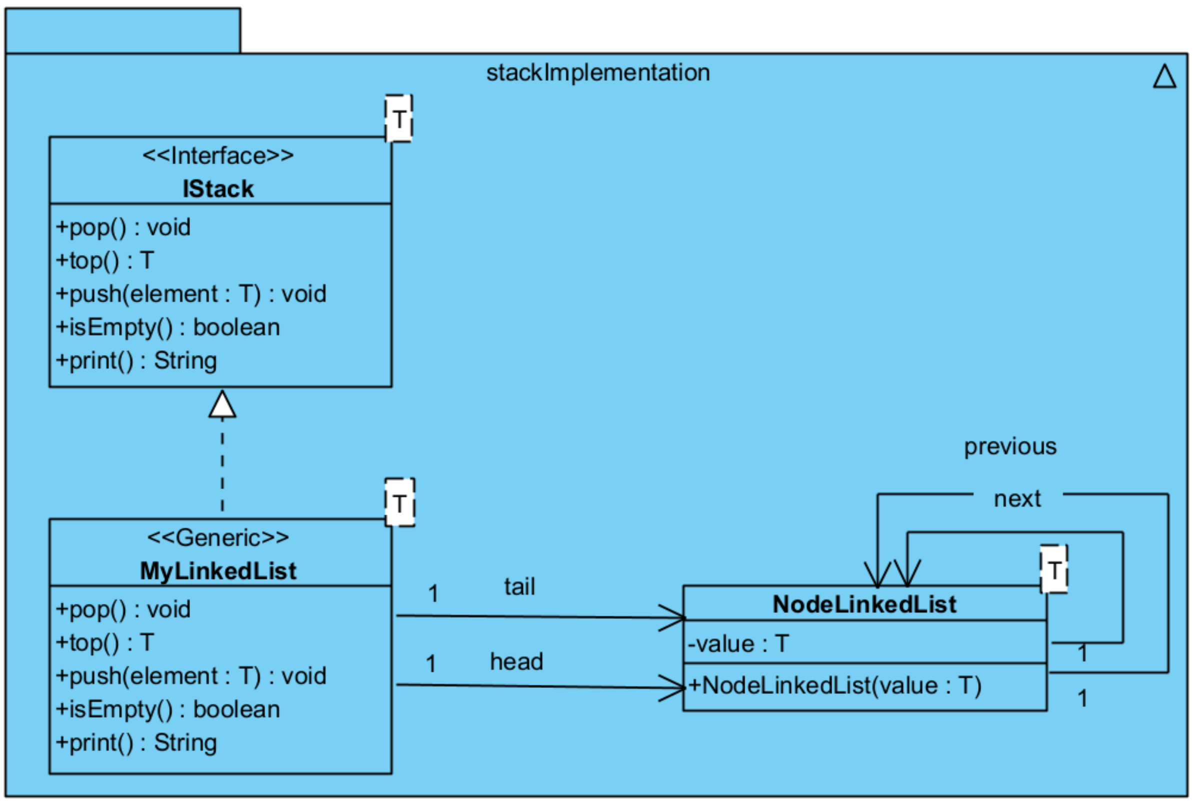
{Pre: hashTable!= null ^ hashTable!= {∅} key ∈ hashTable}
“The hash table must already be initialized, must not be empty and the key must exist in the hash table”

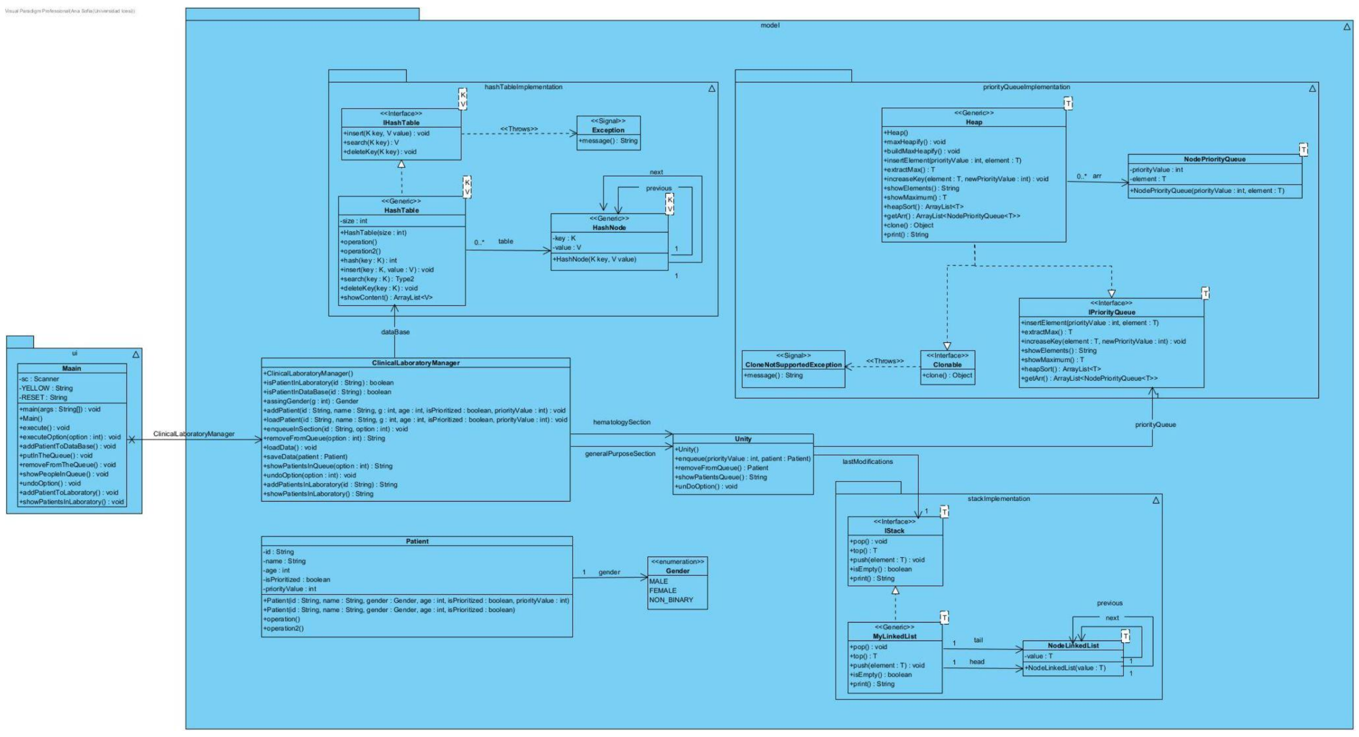
{HashTable = { Element _1 = <””,””>, Element_2 = <””, “”> ... Element_size-1 = <key, element> **} ^ return** Element_size-1 = <key, element> }
“The value of the key has been shown from the hash table”

Design


hashTableImplementation
K
V
<<Interface>>
IHashTable
+insert(K key, V value) : void
+search(K key) : V
+deleteKey(K key) : void
<<Throws>>
<<Signal>>
Exception
+message() : String
K
V
<<Generic>>
HashTable
-size : int
+HashTable(size : int)
+operation()
+operation2()
+hash(key : K) : int
+insert(key : K, value : V) : void
+search(key : K) : Type2
+deleteKey(key : K) : void
+showContent() : ArrayList<V>
0..*
table
next
previous
K
V
<<Generic>>
HashNode
-key : K
-value : V
+HashNode(K key, V value)
1
1


priorityQueueImplementation
T
<<Generic>>
Heap
+Heap()
+maxHeapify() : void
+buildMaxHeapify() : void
+insertElement(priorityValue : int, element : T)
+extractMax() : T
+increaseKey(element : T, newPriorityValue : int) : void
+showElements() : String
+showMaximum() : T
+heapSort() : ArrayList<T>
+getArr() : ArrayList<NodePriorityQueue<T>>
+clone() : Object
+print() : String
0..*
arr
T
NodePriorityQueue
-priorityValue : int
-element : T
+NodePriorityQueue(priorityValue : int, element : T)
T
<<Interface>>
IPriorityQueue
+insertElement(priorityValue : int, element : T)
+extractMax() : T
+increaseKey(element : T, newPriorityValue : int) : void
+showElements() : String
+showMaximum() : T
+heapSort() : ArrayList<T>
+getArr() : ArrayList<NodePriorityQueue<T>>
<<Signal>>
CloneNotSupportedException
+message() : String
<<Throws>>
<<Interface>>
Clonable
+clone() : Object


stackImplementation
T
<<Interface>>
IStack
+pop() : void
+top() : T
+push(element : T) : void
+isEmpty() : boolean
+print() : String
T
<<Generic>>
MyLinkedList
+pop() : void
+top() : T
+push(element : T) : void
+isEmpty() : boolean
+print() : String
1
tail
1
head
previous
next
T
NodeLinkedList
-value : T
+NodeLinkedList(value : T)
1
1


Visual Paradigm Professional(Ana Sofia(Universidad Icesi))
model
hashTableImplementation
<<Interface>>
IHashTable
+insert(K key, V value) : void
+search(K key) : V
+deleteKey(K key) : void
<<Throws>>
<<Signal>>
Exception
+message() : String
<<Generic>>
HashTable
-size : int
+HashTable(size : int)
+operation()
+operation2()
+hash(key : K) : int
+insert(key : K, value : V) : void
+search(key : K) : Type2
+deleteKey(key : K) : void
+showContent() : ArrayList<V>
0..*
table
next
previous
<<Generic>>
HashNode
-key : K
-value : V
+HashNode(K key, V value)
1
1
dataBase
ui
Maain
-sc : Scanner
-YELLOW : String
-RESET : String
+main(args : String[]) : void
+Main()
+execute() : void
+executeOption(option : int) : void
+addPatientToDataBase() : void
+putInTheQueue() : void
+removeFromTheQueue() : void
+showPeopleInQueue() : void
+undoOption() : void
+addPatientToLaboratory() : void
+showPatientsInLaboratory() : void
ClinicalLaboratoryManager
ClinicalLaboratoryManager
+ClinicalLaboratoryManager()
+isPatientInLaboratory(id : String) : boolean
+isPatientInDataBase(id : String) : boolean
+assingGender(g : int) : Gender
+addPatient(id : String, name : String, g : int, age : int, isPrioritized : boolean, priorityValue : int) : void
+loadPatient(id : String, name : String, g : int, age : int, isPrioritized : boolean, priorityValue : int) : void
+enqueueInSection(id : String, option : int) : void
+removeFromQueue(option : int) : String
+loadData() : void
+saveData(patient : Patient)
+showPatientsInQueue(option : int) : String
+undoOption(option : int) : void
+addPatientsInLaboratory(id : String) : String
+showPatientsInLaboratory() : String
hematologySection
generalPurposeSection
Unity
+Unity()
+enqueue(priorityValue : int, patient : Patient)
+removeFromQueue() : Patient
+showPatientsQueue() : String
+unDoOption() : void
priorityQueueImplementation
<<Generic>>
Heap
+Heap()
+maxHeapify() : void
+buildMaxHeapify() : void
+insertElement(priorityValue : int, element : T)
+extractMax() : T
+increaseKey(element : T, newPriorityValue : int) : void
+showElements() : String
+showMaximum() : T
+heapSort() : ArrayList<T>
+getArr() : ArrayList<NodePriorityQueue<T>>
+clone() : Object
+print() : String
0..* arr
NodePriorityQueue
-priorityValue : int
-element : T
+NodePriorityQueue(priorityValue : int, element : T)
<<Interface>>
IPriorityQueue
+insertElement(priorityValue : int, element : T)
+extractMax() : T
+increaseKey(element : T, newPriorityValue : int) : void
+showElements() : String
+showMaximum() : T
+heapSort() : ArrayList<T>
+getArr() : ArrayList<NodePriorityQueue<T>>
<<Signal>>
CloneNotSupportedException
+message() : String
<<Throws>>
<<Interface>>
Clonable
+clone() : Object
priorityQueue
lastModifications
Patient
-id : String
-name : String
-age : int
-isPrioritized : boolean
-priorityValue : int
+Patient(id : String, name : String, gender : Gender, age : int, isPrioritized : boolean, priorityValue : int)
+Patient(id : String, name : String, gender : Gender, age : int, isPrioritized : boolean)
+operation()
+operation2()
1 gender
<<enumeration>>
Gender
MALE
FEMALE
NON_BINARY
stackImplementation
<<Interface>>
IStack
+pop() : void
+top() : T
+push(element : T) : void
+isEmpty() : boolean
+print() : String
<<Generic>>
MyLinkedList
+pop() : void
+top() : T
+push(element : T) : void
+isEmpty() : boolean
+print() : String
1 tail
1 head
previous
next
NodeLinkedList
-value : T
+NodeLinkedList(value : T)

Test cases

Stack:

Scenarios configuration:

Scenario	Class	Scenario
setUp1()	MyLinkedListTest	• No elements added to the stack

setUp2()	MyLinkedListTest	myLinkedList.push(12);
		myLinkedList.push(4);
		myLinkedList.push(5);
		myLinkedList.push(9);
		myLinkedList.push(5);
		myLinkedList.push(69);

Tests:

Test Goal: Check the correct insertion of the elements in the stack

Class	Method	Scenario	Input Values	Result
MyLinkedListTest	Push()	setUp1();	“1”,”2”,”3”	The elements in the stack should have been added in the following order: “1,2,3”
MyLinkedListTest	Push()	setUp1();	“”	There is no current elements in the stack, so it should be printing: “”
MyLinkedListTest	Push()	setUp1();	“12”,”4”,”5”,”9”,”5”,”69”	The elements in the stack should have been added in the following order: “69 5 9 5 4 12”

Test Goal: Check the view of the top element in the stack

Class	Method	Scenario	Input Values	Result
MyLinkedListTest	Top()	setUp2();	“”	The top value of the stack should be “69”
MyLinkedListTest	Top()	setUp1();	“”	The top value of the stack should be null since there are no elements on the stack
MyLinkedListTest	Top()	setUp1();	“13”	The top value of the stack should be “13”

Test Goal: Check the removal of the top element in the stack

Class	Method	Scenario	Input Values	Result
MyLinkedListTest	Pop()	setUp2();	“”	After removing the top of “69”, the next top should be “5”
MyLinkedListTest	Pop()	setUp1();	“”	Sinces there is no elements in the stack, after removing the previous top, the current top should be null

MyLinkedListTest	Pop()	setUp1();	“69”	After removing the recently inserted valued, the top of thos stack should be null

Test Goal: Check if the method correctly evaluates the stack				
Class	Method	Scenario	Input Values	Result
MyLinkedListTest	isEmpty()	setUp1();	“”	Since the stack is empty, it should return TRUE
MyLinkedListTest	isEmpty()	setUp1();	“420”	Since the stack was previously empty and now it has a new element, it should return FALSE
MyLinkedListTest	isEmpty()	setUp1();	“77”	After adding and removing an element, the stack should be empty so it should return TRUE

Priority Queue:

Scenarios configuration:

Scenario	Class	Scenario
setUp1()	HeapTest	● No elements added to the priority queue

setUp2()	HeapTest	insertElement(5, "Juan");
		insertElement(1, "Francisco");
		insertElement(4, "Alberto");
		insertElement(6, "Javier");
		insertElement(9, "Nelson");
		insertElement(11,"Patricia");

setUp3()	HeapTest	getArr().add(16, "Juan"));
		getArr().add((4,"Francisco"));
		getArr().add(10, "Alberto));
		getArr().add(14, "Javier");
		getArr().add(7, "Nelson");
		getArr().add(9, "Patricia");
		getArr().add(3, "Nelly");

		getArr().add(2, "Esteban");
		getArr().add(8, "Julio");
		getArr().add(1, "Andrew");

setUp4()	HeapTest	getArr().add(4, "Juan");
		getArr().add(14,"Francisco");
		getArr().add(7, "Alberto");
		getArr().add(2, "Javier");
		getArr().add(8, "Nelson");
		getArr().add(1, "Patricia");

setUp5()	HeapTest	insertElement(5, "Juan");

setUp6()	HeapTest	insertElement(3, "Ricardo1");
		insertElement(3, "Ricardo2");
		insertElement(3, "Ricardo3");
		insertElement(3, "Ricardo4");
		insertElement(3, "Ricardo5");

setUp7()	HeapTest	insertElement(16, "Juan");
		insertElement(3, "Alfa");
		insertElement(4,"Beta");
		insertElement(1, "Gamma");
		insertElement(3, "Omega");

Tests:

Test Goal: Check the correct insertion of the elements in the priority queue

Class	Method	Scenario	Input Values	Result
Heap	insertElement()	setUp1();	13,"Hola"	There should only one element in the priority queue with key “Hola” and priority value 13
Heap	insertElement()	setUp2();	13,"Hola"	The element “Hola” should be added at the beginning of the queue because it’s the element with the biggest priorityValue
Heap	insertElement()	setUp2();	13,"Hola"	The priority values in the array should be “13 6 11 1 5 4 9”

Test Goal: Check the correct functionality of the method maxHeapify in the priority queue				
Class	Method	Scenario	Input Values	Result
Heap	maxHeapify()	setUp3();	2	The array in the priority queue should have the following order: "16 14 10 8 7 9 3 2 4 1 "
Heap	maxHeapify()	setUp4();	1	The array in the priority queue should have the following order: "14 8 7 2 4 1 "
Heap	maxHeapify()	setUp3();	1	The array in the priority queue should have the following order: "16 4 10 14 7 9 3 2 8 1 "

Test Goal: Check the correct functionality of the method maxHeapify in the priority queue				
Class	Method	Scenario	Input Values	Result
Heap	maxHeapify()	setUp3();	2	The array in the priority queue should have the following order: "16 14 10 8 7 9 3 2 4 1 "
Heap	maxHeapify()	setUp4();	1	The array in the priority queue should have the following order: "14 8 7 2 4 1 "
Heap	maxHeapify()	setUp3();	1	The array in the priority queue should have the following order: "16 4 10 14 7 9 3 2 8 1 "

Hash Table:

Scenarios configuration:

Scenario	Class	Scenario

setUp1()	HashTableTest	• m = 13
setUp2()	HashTableTest	• m = 1

setUp3()	HashTableTest	• m = 1
		
		insert("123", "Juan Jose")
		insert("456", "Patricia")

setUp4()	HashTableTest	• m = 1
		insert("124", "Pepe");
		insert("124", "Papa");

Tests:

Test Goal: Check the generation of the hash function of an element				
Class	Method	Scenario	Input Values	Result
HashTable	table.hash()	setUp1();	“Juan”	The hash value is 4
HashTable	table.hash()	setUp1();	“”	The hash value is 0
HashTable	table.hash()	setUp2();	“SapoPerro”	The hash value is 0

Test Goal: Check the correct insertion in the hash table with and without collisions				
Class	Method	Scenario	Input Values	Result
HashTable	table.insert()	setUp2();	“123", "Juan Jose"	The element should be added to the hash table with key “123” and value “Juan Jose ”
HashTable	table.insert()	setUp2();	"123", "Juan Jose" "456", "Patricia"	There should be 2 elements in the hash table, the first with key “123” and value “Juan Jose ” and the second with key “456” and value “Patricia” having a

				collison
HashTable	table.insert()	setUp1();	"4", "Diaz" "2", "Londoño"	There should be 2 elements added to the hash table with no collisions in them.

Test Goal: Check the correct search in the hash table				
Class	Method	Scenario	Input Values	Result
HashTable	table.search()	setUp3();	"456"	The method should return the value “Patricia”
HashTable	table.search()	setUp3();	"789"	The method should return a null value because in the hash table doesn’t exist an element with that key
HashTable	table.search()	setUp4();	"124"	The method should return the value “Pepe”. There we evaluate what happen when we add 2 elements with the same key

Test Goal: Check the correct removing in the hash table				
Class	Method	Scenario	Input Values	Result
HashTable	table.deleteKey()	setUp3();	"456"	The method should remove the element “Patricia” and conserve the structure of the hash table with collisions
HashTable	table.deleteKey()	setUp2();	table.insert("123	The method should

			", "Juan Jose");
			
			table.insert("456
			", "Patricia");
			
			"table.deleteKey(
			"123");
				remove the element "Juan Jose" and conserve the structure of the hash table without collisions

HashTable	table.deleteKey()	setUp1();	"123"	The method shouldn't erase any object because there isn't an object yet

References

- Sumo Logic, Inc. (2021, 14 octubre). What is a Software Stack? Sumo Logic. Recuperado 21 de octubre de 2022, de https://www.sumologic.com/glossary/software-stack/
- Semilof, M. & Montgomery, J. (2020, 30 noviembre). software stack. SearchAppArchitecture. Recuperado 21 de octubre de 2022, de https://www.techtarget.com/searchapparchitecture/definition/software-stack
- Yang, P. L. |. (2021, 14 diciembre). System Design — Message Queues - Computer Science Fundamentals. Medium. Recuperado 21 de octubre de 2022, de https://medium.com/must-know-computer-science/system-design-message-queues-245612428a22
- Stemmler, K. (2022, 19 enero). Hash Tables | What, Why & How to Use Them | Khalil Stemmler. Recuperado 21 de octubre de 2022, de https://khalilstemmler.com/blogs/data-structures-algorithms/hash-tables/
- Braunschweig, D. (2018, 15 diciembre). Fixed and Dynamic Arrays – Programming Fundamentals. Pressbooks. Recuperado 21 de octubre de 2022, de https://press.rebus.community/programmingfundamentals/chapter/fixed-and-dynamic-arrays/
- Dynamic Array in Java - Javatpoint. (s. f.). www.javatpoint.com. Recuperado 21 de octubre de 2022, de https://www.javatpoint.com/dynamic-array-in-java