Documentation

Mateo Rubio - A00400104
Alejandro Quiñones - A00377013

Engineering Method

Phase 1

Client	CyED professors from ICESI university
User	Any person that wants to manage their assignments (tasks and reminders)
Functional Requirements	**FR1**: Store assignments **FR2**: Modify assignments **FR3**: Remove assignments **FR4**: Show user assignments **FR5**: Manage assignments priorities **FR6:** Undo the actions performed
Problem Context (Identification)	You and your partners were required to develop a task and reminder management system that allows users to add, organize and manage their to-do's tasks and reminders.
No Functional Requirements	**NFR1:** Use Hash to store the assignments **NFR2:** Develop a user interface **NFR3**: Use a queues to organize assignments

Identifier and Name	*[FR1 - Store assignments]*		
Summary	*The system allows the user to store assignments and each assignment has the following information: a title, a description, a deadline, a priority, etc.*		
Inputs	**Input Name**	**Data Type**	**Valid Values**
	title	String	
	description	String	
	deadline	Date	Dates prior to current date
	priority	short	
Post-Condition	The assignment is successful stored if all the information is correct, updating the data, also the assignment get a id generate by the system		
Outputs	**Output Name**	**Data Type**	**Format**
	msg_success	String	“New assignment added”
	msg_fail	String	“Something wrong”

Identifier and Name	*[FR2 - Modify assignments]*		
Summary	*The system allow the user modify an assignment, the user can change anything he want, title, description, deadline or priority*		
Inputs	**Input Name**	**Data Type**	**Valid Values**
	title	String	
	description	String	

	deadline	Date	Dates prior to current date
	priority	short	
Post-Condition	The assignments are modified, if the assignment has priority the list get sorted to maintain the order.		
Outputs	**Output Name**	**Data Type**	**Format**
	msg_success	String	“Assignment modified”
	msg_fail	String	“Something wrong”

Identifier and Name	*[FR3 - Remove assignments]*		
Summary	*The system allows the user to remove an assignment if he wants.*		
Inputs	**Input Name**	**Data Type**	**Valid Values**
	title	String	
	id	String	
Post-Condition	The assignment is successful remove		
Outputs	**Output Name**	**Data Type**	**Format**
	msg_success	String	“Assignment removed”
	msg_fail	String	“Something wrong”

Identifier and Name	*[FR4 - Show user assignments]*		
Summary	*The system must show the user assignments by the selected filter (deadline, arrival or priority)*		
Inputs	**Input Name**	**Data Type**	**Valid Values**
	type	Priority/Date	Not null elements
Post-Condition	The assignment is successful stored		
Outputs	**Output Name**	**Data Type**	**Format**

	tasks	String	Task1:{each property}
			Task2:{each property}

	msg_fail	String	“Something wrong”

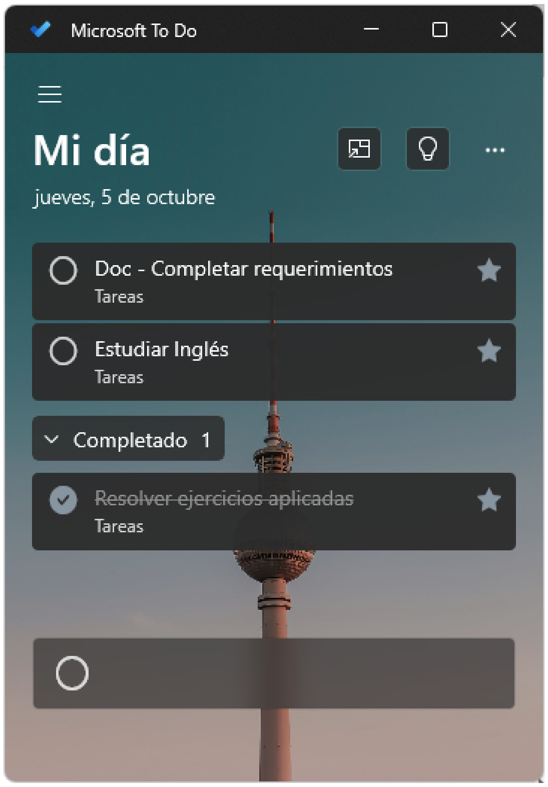
Identifier and Name	*[FR5 - Manage assignments priorities]*		
Summary	*The system allows the user to classify their assignments in 2 main branches priority and non priority* *The first branch is organized first based on the priority level and then the deadline of the assignment, while the non priority is organized in the same way which the assignments were stored on the system*		
Inputs	**Input Name**	**Data Type**	**Valid Values**
	assignments	HashTable	*“Asignments already stored, present in the hash table”*

Post-Condition	If there is any assignment(s) then two queues are created in order to ease the process of visualizing what is pending, this two queues correspond to priority and non priority assignments, the first one also entails a sort mechanism		
Outputs	**Output Name**	**Data Type**	**Format**
	priorityAsgQueue	Priority Queue	"linked queue"
	nonPrioQueue	Queue	"linked queue"

Identifier and Name	*[FR6 - Undo the actions performed]*		
Summary	*The system reverts/undo the action done just before returning the data to its previous status,* *For this an action must be done before else there will be nothing to undo.*		
Inputs	**Input Name**	**Data Type**	**Valid Values**
	None	N/A	N/A
Post-Condition	If the precondition is fulfilled then the data affected by the just done action is returned to its previous state.		
Outputs	**Output Name**	**Data Type**	**Format**
	assignments	HashTable	
	priorityAsgQueue	Priority Queue	"linked queue"
	nonPrioQueue	Queue	"linked queue"

<u>Phase 2:</u>

Precedents: We did research about apps that worked similar to the one we should develop, and found, first Microsoft To-Do which allows us to store and visualize our activities and order them as we want. Then, keep notes by google is a complex application to add notes and reminders that may pop depending on how you configure them, the notes not necessarily are ordered, but you can classify them in labels or groups based on your own criteria.


Microsoft To Do

Mi día

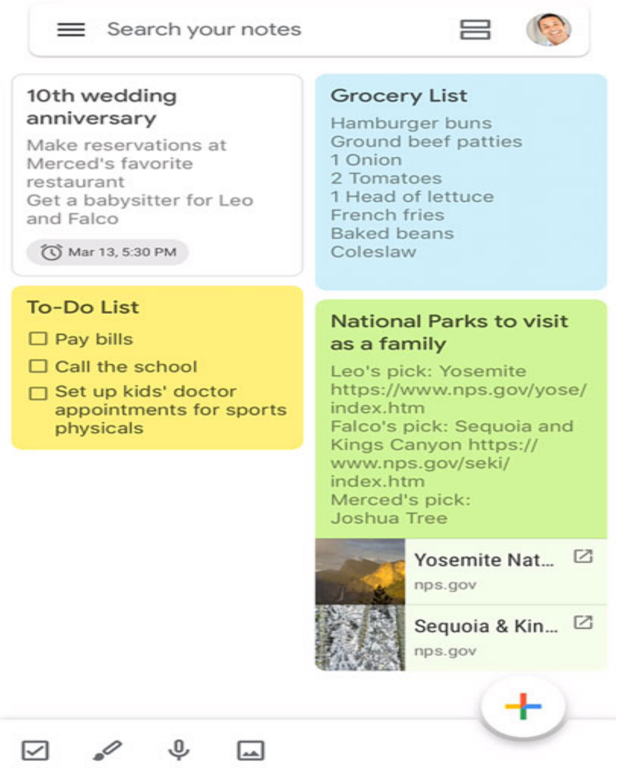
jueves, 5 de octubre

○ Doc - Completar requerimientos
Tareas

○ Estudiar Inglés
Tareas

Completado 1

✓ ~~Resolver ejercicios aplicadas~~
Tareas


Search your notes

10th wedding anniversary
Make reservations at Merced's favorite restaurant
Get a babysitter for Leo and Falco
Mar 13, 5:30 PM

Grocery List
Hamburger buns
Ground beef patties
1 Onion
2 Tomatoes
1 Head of lettuce
French fries
Baked beans
Coleslaw

To-Do List
□ Pay bills
□ Call the school
□ Set up kids' doctor appointments for sports physicals

National Parks to visit as a family
Leo's pick: Yosemite https://www.nps.gov/yose/index.htm
Falco's pick: Sequoia and Kings Canyon https://www.nps.gov/seki/index.htm
Merced's pick: Joshua Tree

Yosemite Nat...
nps.gov

Sequoia & Kin...
nps.gov

To get all the information needed we compared our perspectives after reading the document that tags all the main point of the application, which in this case is all we got from the “Client” and also check and specify certain pieces of data that weren’t clear enough for the team, such as:

- Ordering criteria, as is just defined as something, so we thinked of a simple number scale determined by the user
- Undo actions till which point, this to understand how much could be done in the future method, and to analyze this type of application and if they use this type of actions, or compare them to simpler pc apps that allow Ctrl + Z.

Phase 3:

a) Brainstorm

+ Maybe use Java Swing for the interface, we kind of discuss the interface we will use so also we got informed on some of them and check the value each of them could provide to us.
+ Use heapsort to sort priority assignments, as we have to order some assignments, we listed the type of sort we could use like insertion, selection or heapsort, but due to the temporal complexity and the composition of the priority queue we decided heapsort was optimal.
+ Solve collisions on the hash table, we thought mostly about redirection and chaining as those methods are very tempting, but as redirection can lead to fulling a hashtable we decided chaining for large amounts of data.
+ Undo action can be made with the opposite action the user makes (in code with opposite functions) and just save temp variables, at first we thought it will be a method only but then we realize as it's a stack we need to push some sort of thing or way to identify the last action, and enum seem tempting but it may be too much for an action also there is the fact that we also need to save a temporal value, it could be the whole or part of the list of total items or just the item affected identified by the title.
+ To hash the assignments maybe use the title and description and use the value each string represents to module it in the function hash, it may reduce the possibility of collisions, but as we know also the user can modify attributes of the assignment so it may be complicated to locate something if the things we are using to hash it are being changed, specially if it was modify and then tried to undo the action, so we decided to select a id that couldn't be changed.

b) Review list.

+ How to hash the assignments
+ Which framework or how to made the interface
+ how to sort the priority assignments
+ how to solve the coalition in the hash table
+ how to made the undo action
+ The way to show the info

Phase 4 (View Phase 5 to understand de points)

A) As it was told before, heapsort was our selection but we thought also to use insertion for other simpler things, as constantly we are adding elements to the priority tue heapsort is the method that fits better with it and also more efficient than others but as we also order things with date, simpler value we prefer sorting it by insertion which is simpler to apply and does not require a preset. [UND (7 pts), EFF (3 pts), ADA (8 pts) & FUN (14 pts)].

B) Collisions in our hash will be solved by chaining due to what we said before and also to the fact that we don't know how many assignments could be added to the system, even though it requires enlarging our implementation with a linked list, the benefit afterwards is much much better. [UND (9 pts), EFF (4 pts), ADA (10 pts) & FUN (14 pts)]

C) For the undo functionality we prefer to create a new class, that may receive the type of action done, via Int or Identifier, and the assignment affected and in order to preserve the same hash, we didn't create a inner id but decided hash it just with the title, so it be final, taking into account the type of application the title is just enough to decide whether an assignment is unique or not. [UND (9 pts), EFF (5 pts), ADA (9 pts) & FUN (12 pts)]

D) We also discarded the idea of keeping and updating queues as priority may cause problems when trying to eliminate an assignment in between the queue, also the updating is a constant cost, so we decided just building queues when it is asked to show either of the queues so there is no problem deleting any assignment. [UND (8 pts), EFF (3 pts), ADA (6 pts) & FUN (12 pts)]

Phase 5

The criteria taking into account for the decisions made before were:

+ Understanding, how complex it is to comprehend methods/implementations (1 to 10)
+ Efficiency, in terms of time and resources how much it consumes (1 to 5)

- Adaptability, Given the possibility to add new things to the system how easy they will fit in our code (1 to 10)
- Functionality, related to the client requests how well does it accomplish or approaches to what is asked for (1 to 15)

Test Cases

Name	Class	Setup
testIntInsert	PriorityQueue	Declaring a Int priority queue, inserting some numbers, checking the correct size and insertion after ordering them
testEmpty		Declaring a Int priority queue, checking the front and back value when it is empty, adding one item then extracting it and checking the state of the queue.
testExtracting		Declaring a String priority queue, and inserting some values in priority disorder, then extracting the maximum, and then checking if it is ordered again.
testRandomIntAdd		Declaring int priority queue and inserting 100 random numbers also followed by a max system for then to check if the ordering method works
testStringInsert	Queue	Declaring a String queue and enqueue some elements to check if they are added correctly and follow the FIFO principle
testEmpty		Checking the state of the queue when it is empty, and after adding and deleting elements check if it updated the queue and its size.

testFunctionalities		Checking the peek and size functionality when added a series of elements
testPushPop	Stack	Pushing and popping multiple elements on the stack and checking the state of it and if the LIFO principle is kept
testEmpty		Reviewing the state of an empty stack and what remains after entering elements and delete them till empty again.
testTop		Pushing and popping multiple elements on the stack and evaluating the top functionality series of operations.
testClean		Pushing 100 ordered elements and reviewing them by the LIFO principle popping them one by one.
addTest	HashTable	Declaring a String, Integer Hash Table and validate if is create empty and add elements validating the length
removeTest		Add a element remove them and remove again to validate if the method return -1 length
getTest		Insert a element and use the method get to validate the existence of the element and also try use the method with a nonexistent element
isEmptyTest		Create the String, Integer HashTable and use the method isEmpty to validate if the hash table is empty and after add a element and validate again
sizeTest		Declaring a String, Integer Hash Table and push elements and at the same time validate the length, after remove elements and validate the length on each iteration

Class	Method	Setup	Inputs	Result
PriorityQueue	PriorityQueue() <Integer>	testIntInsert		True, it inserts the inputs correctly so the stack is not empty. The heapsize correspond to the size of the elements minus 1 The front and the back of the queue are as expected the least “important” to the most.
	Insert()		1, 2 , 3	
	isEmpty()			
	heapSort()			
	front()			
	getHeapSize()			
	back()			
	PriorityQueue() <Integer>	testEmpty		True, the esrucure is initialized correctly and when it gets extracted all the elements it returns to an empty state correctly.
	isEmpty()			
	back()			
	insert()		1	
	extratctMax()			
	front()			
	PriorityQueue() <String>	testExtracting		True, after the elements been pushed the max value “U” is excracted then the array is ordered and the max value “S” is were it is supposed to be
	insert()		“R”, “P”, “S”, “A” and “U”	
	front()			
	exctractMax()			
	heapSort()			
	back()			

	PriorityQueue() <Integer>	testRandomInt Add		True, all the numbers are generated, pushed and compared to the previous one to check for the highest and after sorting the queue, the maximum value is were it is supposed to be
	insert()		100 random numbers from 0 to 73	
	isEmpty()			
	heapSort()			
	back()			

Class	Method	Setup	Inputs	Result
Queue	Queue() <String>	testStringInsert		True, the items are added correctly to the queue (evaluated by the size) and follow the FIFO principle
	enqueue()		“Luis”, “Manrique” and “Felipe	
	isEmpty()			
	size()			
	peek()			
	Queue() <Integer>	testEmpty		True, the queue is initialized correctly as at first it is empty and peek return null, after enqueue and dequeue of the 13, 31 and 17, the queue return to an empty state with size 0
	isEmpty()			
	peek()			
	enqueue()		13, 31, 17	

	dequeue()			
	size()			
	Queue() <Integer>	testFunctionalities		True, after adding 5 elements the FIFO is preserved and the peek function always returns the first entered value.
	enqueue()		2,1,4,1,3	
	size()			
	peek()			

Class	Method	Setup	Inputs	Result
Stack	Stack() <String>	testPushPop		True, these 3 elements are pushed correctly into the stack and by popping them one by one we assure the LIFO principle is kept
	push()		"Felony", "Misdemeanor and "Infraction"	
	isEmpty()			
	pop()			
	Stack() <Integer>	testEmpty		True, when created the stack is empy and after the enqueue, check and dequeue of an item the stack returns to an empty state in which top returns null.
	isEmpty()			
	top()			
	push()		1	
	pop()			
	Stack() <Integer>	testTop		True,various elements are pushed and checked by the top function which aways return the last in value, so it be in order with the
	push()		2,4 - 3,5	
	top()			

	pop()			stack logic, which is the cased, even after turning the stack empty after some elements introduced
	Stack() <Integer>	testClean		True, all the elements are correctly pushed to the stack check by the fact it is not empty eany more and the top value is 50, then 51 elements are popped, in the correct order as they correspond to the largest positive to 0, and finally the top value end up being -1, following the LIFO principle
	push()		integers from -50 to 50	
	top()			
	pop()			
	isEmpty()			

Class	Method	Setup	Inputs	Result
HashTable	HashTable() <String, Integer>	addTest		True, the hash table is initialized correctly and length is 0, after add elements the length increase in each iteration and values are 1, 2 and 3
	add()		“key,1”, “key,2” and “key2,2”	
	get()			
	containsKey()			
	remove()			
	length()	removeTest		True, the hash table is initialized correctly and length is 0, after add add element with the
	getHash()			

	values()			key “key” and value 1, after check the length and equals to 1, after remove and check the length and is 0, after remove again to check no get -1
	isEmpty()		“key,1”,”key”,”key”	
	clear()			
		getTest		True, after add the element and use get method for validate the value of the key, after validate inexistence of a key
			“key,1”	
		isEmptyTest	“key,1”	True, start validating if the hash table is empty and returns true, after add and element and check if is empty in this case returns false
		sizeTest	“key,1”,”key,2”,”key2,2”,”key”,”key”,”key”	True, start checking if the length is 0 and return true, after add a element and check the length again but in this case is 1, do the same again and when check length is 2, do the same again and when check length is 3, now remove elements and check the length in each iteration and see how this decrease of 3 to 0

Temporal complexity Analysis

Show content in a Hash Table algorithm:

Statement	Effort

if (tasks.empty())	1
StringBuilder stringBuilder = new StringBuilder()	1
for (Assignment assignment : task.values())	n + (j * k) + 4
stringBuilder.append(assignment);	1
return stringBuilder.toString	1

T(A) = 1 + 1 + n + (j * k) + 4 + 1 + 1
T(A) = n + (j * k) + 8
With this we can say that the time complexity of this algorithm in Big O notation would be O(n * k)

Get values in a Hash Table algorithm

Statement	Effort
ArrayList<V> values = new ArrayList<>()	1
for (HashNode<K, V> node : table)	j
while (node != null)	k
values.add(node.getValue());	1
node = node.getNext();	1
return values;	1

T(A) = 1 + j * k + 1 + 1 + 1
T(A) = (j * k) + 4

With this we can say that the time complexity of this algorithm in Big O notation would be O(n * k)

Spatial complexity Analysis

Modify menu

Statement

```
public static int modifyMenu() {
        System.out.println("1. Modify description");
        System.out.println("2. Modify due date, remember the format yyyy-[m]m-[d]d hh:mm:ss");
        System.out.println("3. Modify priority");
        System.out.println("4. Modify type");
        int opt = input.nextInt();
        input.nextLine();
        if (opt < 1 || opt > 4){
            System.out.println("Invalid option");
            modifyMenu();
        }
        return opt;
}
```

Type	Variable	Length	Amount Values
Input	opt	32	1
Aux	none	-	-
Output	int	32	1

input + aux + output = 2 = O(1)

The spatial complexity of this algorithm is O(1)

Show priority assignments by date

Statement

```
public String showPriorityAssignmentsByDate() {
        buildPriorityAssignments();
        if (!priorityAssignments.isEmpty()) {
            ArrayList<Assignment> aux;
            aux = priorityAssignments.getElements();
            StringBuilder stringBuilder = new StringBuilder();
            for (Assignment assignment : sortByDate(aux)) {
                stringBuilder.append(assignment);
            }
            priorityAssignments = new PriorityQueue<>();
            return stringBuilder.toString();
        }
        priorityAssignments = new PriorityQueue<>();
        return "There are no priority assignments";
    }
```

Type	Variables	Length	Amount Values
Input	none	-	-
Aux	aux	n	1
Output	String	-	0

input + aux + output = n = O(n)
The spatial complexity of this algorithm is O(n)