Jhon Stiven Arboleda, Sebastián Barrera, Alejandro García

Tarea Integradora 1

1. **Identification of the problem**

A large bank wants to develop software that models the operation of one of its offices with the highest flow of people. Its main need is to be able to fulfill all the client's needs.

The bank needs:

* Manage shifts by entering the lines either the customer line or a line with different priorities.
* Manage data tables with all customer information.
* Allow the customer to perform different operations at the time of serving, that is, when their turn comes.

Problem: The bank has no way to handle these types of actions effectively in a software for a high flow of problems.

1. **Research**

**Functional requirements:**

|  |  |
| --- | --- |
| **Name** | **FR1:** Register a customer |
| **Summary** | The system must register a client at the time of obtaining their turn with their respective data |
| **Input** | **name** of the client and **identification** of the client |
| **Output** | The client has been successfully registered |

|  |  |
| --- | --- |
| **Name** | **FR2:** Assign to a row |
| **Summary** | The system must register a client at the time of obtaining their turn with their respective data |
| **Input** | **name** of the client and **identification** of the client |
| **Output** | The client has been successfully registered |

|  |  |
| --- | --- |
| **Name** | **FR3:** Search the database |
| **Summary** | The person in charge of the attention will be able to look for the client in the database with his identity card before the client arrives at his office |
| **Input** | **Identification card** of the client |
| **Output** | Table with customer data. Such as name, identity card, bank account, debit / credit cards, date of payment of the credit card and date it was incorporated into the bank |

|  |  |
| --- | --- |
| **Name** | **FR4:** Withdrawals |
| **Summary** | The client may modify the amount of his savings account when requesting a withdrawal |
| **Input** | **Bank account** of the client |
| **Output** | The customer has made a withdrawal from the account |

|  |  |
| --- | --- |
| **Name** | **FR5:** Consignment |
| **Summary** | The client may modify the amount of his savings account when requesting a consignment. |
| **Input** | **Bank account** of the client |
| **Output** | The customer has made a transfer to the account. |

|  |  |
| --- | --- |
| **Name** | **FR6:** Cancel account |
| **Summary** | Deletes your information from the customer database and adds them to a database exclusively for those who cancel their accounts at the bank. In any case, both the date and the reason for cancellation will be saved. |
| **Input** | **Bank account** of the client |
| **Output** | The account has been deleted |

|  |  |
| --- | --- |
| **Name** | **FR7:** Card payment |
| **Summary** | The user can pay the amount used with the credit card so far.  You can make the payment in cash or through your savings account.  Person can only pay the full card or the fee |
| **Input** | **Bank account** of the client |
| **Output** | The amount of money has been paid. |

|  |  |
| --- | --- |
| **Name** | **FR8:** Perform *undo* |
| **Summary** | It will serve to undo mistakes, even after they have been saved |
| **Input** | **----------** |
| **Output** | The action has been undone. |

|  |  |
| --- | --- |
| **Name** | **FR9:** Perform *redo* |
| **Summary** | It will serve to redo mistakes, even after they have been saved |
| **Input** | **----------** |
| **Output** | The action has been redone. |

|  |  |
| --- | --- |
| **Name** | **FR10:** Create savings account |
| **Summary** | The user is created a savings account only if he is not a customer of the bank |
| **Input** | **Name** of the client, **id** of the client, **gender** of the client, **age** of the client, **pregnated, owe** of the card, **cardSpace, quotas, fees, paymentDay** |
| **Output** | The account was created |

|  |  |
| --- | --- |
| **Name** | **FR11:** Create card |
| **Summary** | The user is created a card only if he is not a customer of the bank |
| **Input** | **Name** of the client, **id** of the client, **gender** of the client, **age** of the client, **pregnated,** |
| **Output** | The action has been redone. |

**Stacks:** Stacks are data structures wherein the last element that enters is the first element that leaves (LIFO, **L**ast **I**n **F**irst **O**ut).

**Queues:** Queues are data structures wherein the first elements that enters is the first element that leaves (FIFO, **F**irst **I**n **F**irst **O**ut)

**Hash Tables:** Hash tables are data structures that map keys to values. In the Java programming language, keys and values can be any object that is not null. The hash tables use a hash function to map the key to the value. This means that to retrieve a value the user must enter the key associated with that value.

A problem that might arise with hash tables are collisions, which happen when a given key has more than 2 values associated with it. These problems are solved with open addressing and chaining. In open addressing, the second value is stored in another address. In chaining, both values are stored in a linked list.

**Heap:** A heap is a tree-like data structure with information belonging to an ordered set. Maximum mounds have the characteristic that each parent node has a value greater than that of any of its child nodes, while in minimum mounds, the value of the parent node is always less than that of its child nodes. A tree satisfies the heap condition if it satisfies the previous condition and is also a nearly complete binary tree. A binary tree is complete when all levels are full, with the exception of the last one, which is filled from left to right.

**Bank:** Attention in a bank generally works as follows: There are two lines, general users and priority users (usually customers with a preferential account or people with disabilities). To assign the clients to a line and give them their turn to be attended, they must register with their name and ID, the correspondent in the cubicle will obtain the information from the client and will be able to carry out the operations that the client needs (withdraw, consign, cancel the account and / or pay an amount)

In the event that the client does not have an account within the bank, he or she may create it once it is his turn to be served.

1. **Creative solutions**

**Brainstorming:**

* 1. **Keep a manual record of users:** Each time a user enters, they will have to manually register in a book and thus an advisor will assign them a shift.
  2. **Hire third-party software that fulfill its functions:** The bank will buy the software from a company outside the same bank. The software will have features similar to those you need.
  3. **Create a software:** Hire a software development company to make software to the bank with the essential needs they need.
  4. **Have an online database:** Customers can register in the database online and managers in each cubicle can review the information in the same way.

1. **Selection of the best solution**
   1. **Keep a manual record of users:** It´s inefficient and slow for a high flow of people. **It´s not a good solution.**
   2. **Hire third-party software that fulfill its functions:** It is expensive to acquire software from another company and it is not guaranteed that all needs will be met. **It´s not a good solution**
   3. **Create a software:** It is a good option, because you can have a suitable software for the bank, meeting the exact needs that they require. **It´s a good solution.**
   4. **Have an online database:** When the bank does not have internet, the database will not be accessible and attention will be lost. **It´s not a good solution**

Final decision: CREATE A SOFTWARE.

1. **Diagrama de clases**

**Test diagram de clase**

**ADT (Abstract Data Type)**

**Test Case**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scene** | **Input** | **Result** |
| Queue |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scene** | **Input** | **Result** |
| Stack |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scene** | **Input** | **Result** |
| Hash Table |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scene** | **Input** | **Result** |
| Heap |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scene** | **Input** | **Result** |
| PriorityQueue |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |