ENGINEERING METHOD

<u>PARTICIPANTS</u>

CAMILO CAMPAZ JIMÉNEZ

DANIEL ESTEBAN JARABA GAVIRIA

JOHAN STIVEN RICARDO SIBAJA

TEACHER

URAM ANIBAL SOSA

INTEGRATIVE TASK #1

ALGORITHMS AND DATA STRUCTURES

2021-02

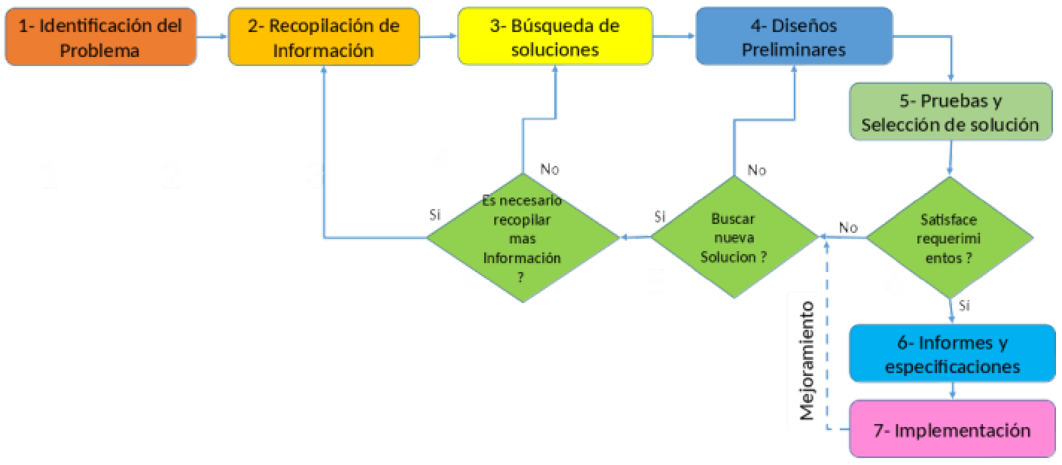
ENGINEERING METHOD

PROBLEM CONTEXT:

A foreign millionaire has decided to undertake in the city of Cali. He wants to start a video game store that provides his services in an innovative way, that is why he needs the development of a software tool capable of clearly and easily simulating the operation of the new video game store that is planned to open in the city is required.

SOLUTION DEVELOPMENT:

To solve the previous situation, the Engineering Method was chosen to develop the solution following a systematic approach and in accordance with the problematic situation posed.
Based on the description of the Engineering Method in the book "Introduction to Engineering" by Paul Wright.
The following flow chart was defined, the steps of which we will follow in the development of the solution.


1- Identificación del Problema
2- Recopilación de Información
3- Búsqueda de soluciones
4- Diseños Preliminares
5- Pruebas y Selección de solución
Satisface requerimi entos ?
No
Buscar nueva Solucion ?
No
Si
Es necesario recopilar mas Información ?
No
Si
Si
6- Informes y especificaciones
7- Implementación
Mejoramiento

IDENTIFICATION OF THE PROBLEM:

In this section, it is important to have a good definition of what the solution to the problem that is being addressed is necessary to contain.
Identification of needs:

- Tool which allows to simulate the operation of the new store that will be put at the service of citizens.
- The tool must be able to receive test data to make the respective demonstration of the operation of the service.
- The tool needs to provide a report with the customers' exit order as well as including the value of each purchase and the order in which their games were packaged.

- The tool must use in its system sorting algorithms with temporal complexity equal to or less than $O(n^2)$
- The tool must use data structures (queues, stacks, hash tables)

INFORMATION GATHERING:

In this section the concepts that are essential for understanding the solution of the specific problem are specified.

Algorithm:

An ordered set of systematic operations that allows making a calculation and finding the solution to a type of problem.

Sort algorithm:

In computation and mathematics, an ordering algorithm is an algorithm that puts elements of a list or a vector in a sequence given by an order relation, that is, the output result must be a permutation —or reordering— of the input that satisfies the given order relation.

Data structures:

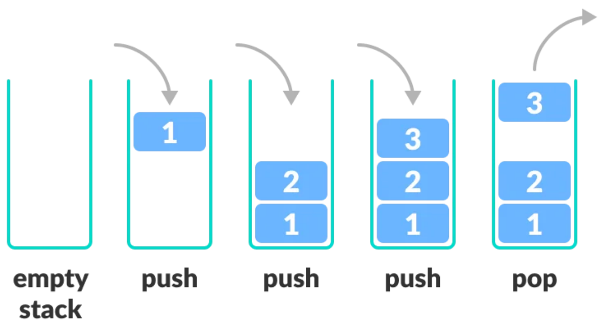
In computer science, a data structure is a particular way of organizing data in a computer so that it can be used efficiently. Different types of data structures are suitable for different types of applications, and some are highly specialized for specific tasks.

Temporal complexity:

In computing, time complexity is computational complexity that describes the amount of time it takes to run an algorithm. ... Therefore, the amount of time required and the number of elementary operations performed by the algorithm differ by a constant factor at most.

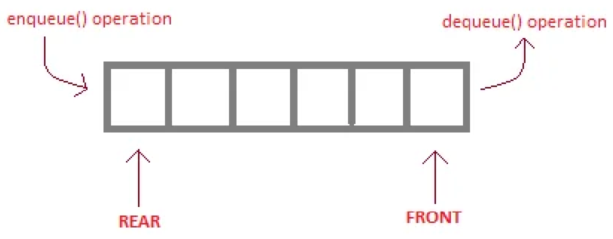
Stack:

A stack is an abstract data type that holds an ordered, linear sequence of items. In contrast to a queue, a stack is a last in, first out (LIFO) structure. A real-life example is a stack of plates: you can only take a plate from the top of the stack, and you can only add a plate to the top of the stack.



Queue:

A Queue is a linear structure which follows a particular order in which the operations are performed. The order is First In First Out (FIFO)

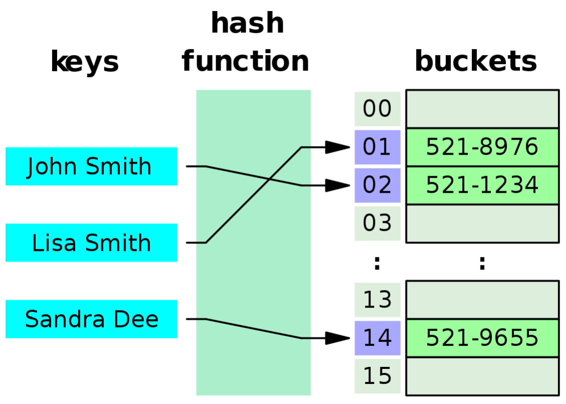


enqueue( ) is the operation for adding an element into Queue.
dequeue( ) is the operation for removing an element from Queue .

QUEUE DATA STRUCTURE

Hash table:

In computing, a hash table (hash map) is a data structure that implements an associative array abstract data type, a structure that can map keys to values. A hash table uses a hash function to compute an index, also called a hash code, into an array of buckets or slots, from which the desired value can be found. During lookup, the key is hashed and the resulting hash indicates where the corresponding value is stored.



SEARCH FOR CREATIVE SOLUTIONS

In this section is going to be introduced the ideas that can be optimal to solve the problem that is proposed.

First things first, the method for the data reading it's pretty because it uses the classical scanner or buffer, so it is not necessary to think about it deeply.

Proposal 1:

In this case it is important to describe the problem as general as possible because the solution might be optimal.

Moreover, thinking about the way to save and move the data in this proposal the idea is to take the principal components of the problem (*Games, customers, etc*), for example, in the shelving the games will have a code and a price, so it is easy to think that a hash table will let know easily the game that the client wants. When the client select the games that he wants to buy we are going to store them in a stack that let us to organize them like in a real situation, furthermore when the clients be ready to pay an efficient way to represent that is with a queue data structure because it will allow us to access quickly to the person that might be attended (*In constant time O(1)*), also, if we want to add more clients to the queue it might be constant too.

Thinking about the output of the solution, it should have the ID of the client, total price, and the codes of all the games he bought.

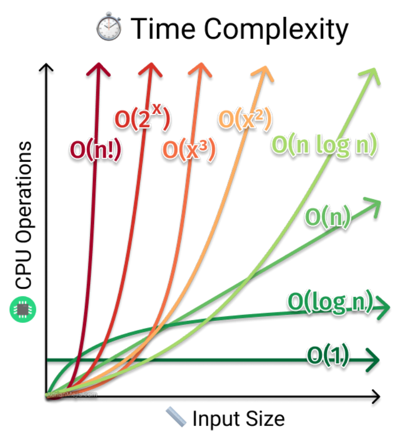
Proposal 2:

This second proposal thinks about working all the data structures as simple arrays, in other words, simplify the way that we save data, in this case is easier to know for example which game is where knowing his position, the same for the pay queue, organize the clientes in a simple array where always takes the first index of the array an then moves to the left the others. It's a pretty simple idea but has a problem, the time complexity because the for example, the access to an index in the array is constant (O(1)), sadly in the way to insert time is O(n) in the worst case, also delete is O(n)

TRANSITION FROM FORMULATION OF IDEAS TO PRELIMINARY DESIGNS

In this case it is considered that the best option is the proposal 1, however it is important to see why the other proposal is not that efficient.

As we can see, the principal difference between the proposals are the time complexity if we think that the algorithm will be the same, just changing the data structure that we are going to use, so in that case the proposal 2 is worst in the fact that in the pay and store game cases generally the complexity that we are going to work with is O(n), linear so it the depends of the size of the input. We don't want that, as we mentioned before we need to think about the problem in general, so in this case the proposal gives us the major part of the algorithm in constant time when we talk about the data structures that we are going to use.



As we can see in the graphic the best complexity is O(1) in other words constant

EVALUATION AND SELECTION OF THE BEST SOLUTION

Is time to select the final choice, so it is important to define evaluation criteria that let us prove that the solution we select is optimal (for us) for this problem, in particular thinking about it generally.

Criteria A: Solve the problem correctly

- [2] Yes
- [1] No

Criteria B: The data structures that use let storage data naturally (it means to don't need to create another structure to save in)

- [3] Yes
- [2] Sometimes
- [1] No

Criteria C: The data structures that use the most are optimal (It means that for example the most are O(1))

- [3] Yes
- [2] More or less
- [1] no

	Criteria A	***Criteria B***	***Criteria C***	***Total***
Proposal 1	2	3	3	8
Proposal 2	2	2	2	6

REPORT PREPARATION AND SPECIFICATIONS

Don't forget the limitations of the solution, so it is important to explain which are the limitations of the solution or considerations for the implementation.

In this case is needed show how will be the solution that is proposed

Considerations:

1. For the optimal work in the data structures it might be all the data in the correct format
2. All the data structures are might be defined properly
3. To create properly the objects all the data that is given might be correct
4. To find a game with the code the game have to exist

In case of console input the process of the solution will look like this:

DESIGN IMPLEMENTATION

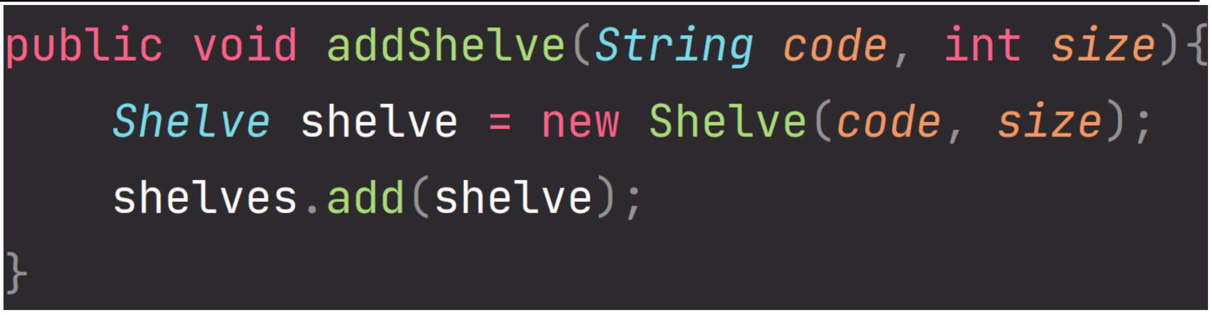
List of tasks to implement:

a. Add shelves
b. Add games to shelves
c. Add clients
d. Find games in shelves

Specifications and subroutines:

A:

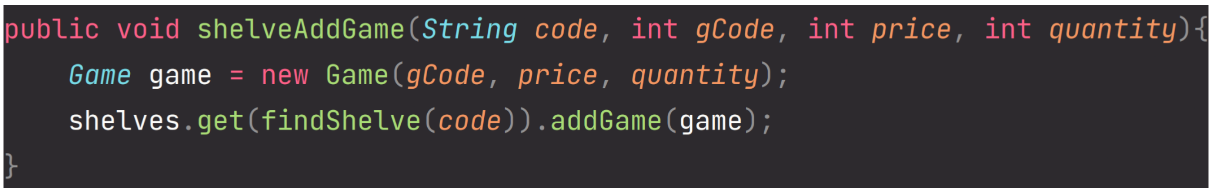
Name:	addShelve
Description:	Create a new shelve
Input:	- code: The code of the shelve - size: The size of the shelve
Output:	



```java
public void addShelve(String code, int size){
    Shelve shelve = new Shelve(code, size);
    shelves.add(shelve);
}
```

B:

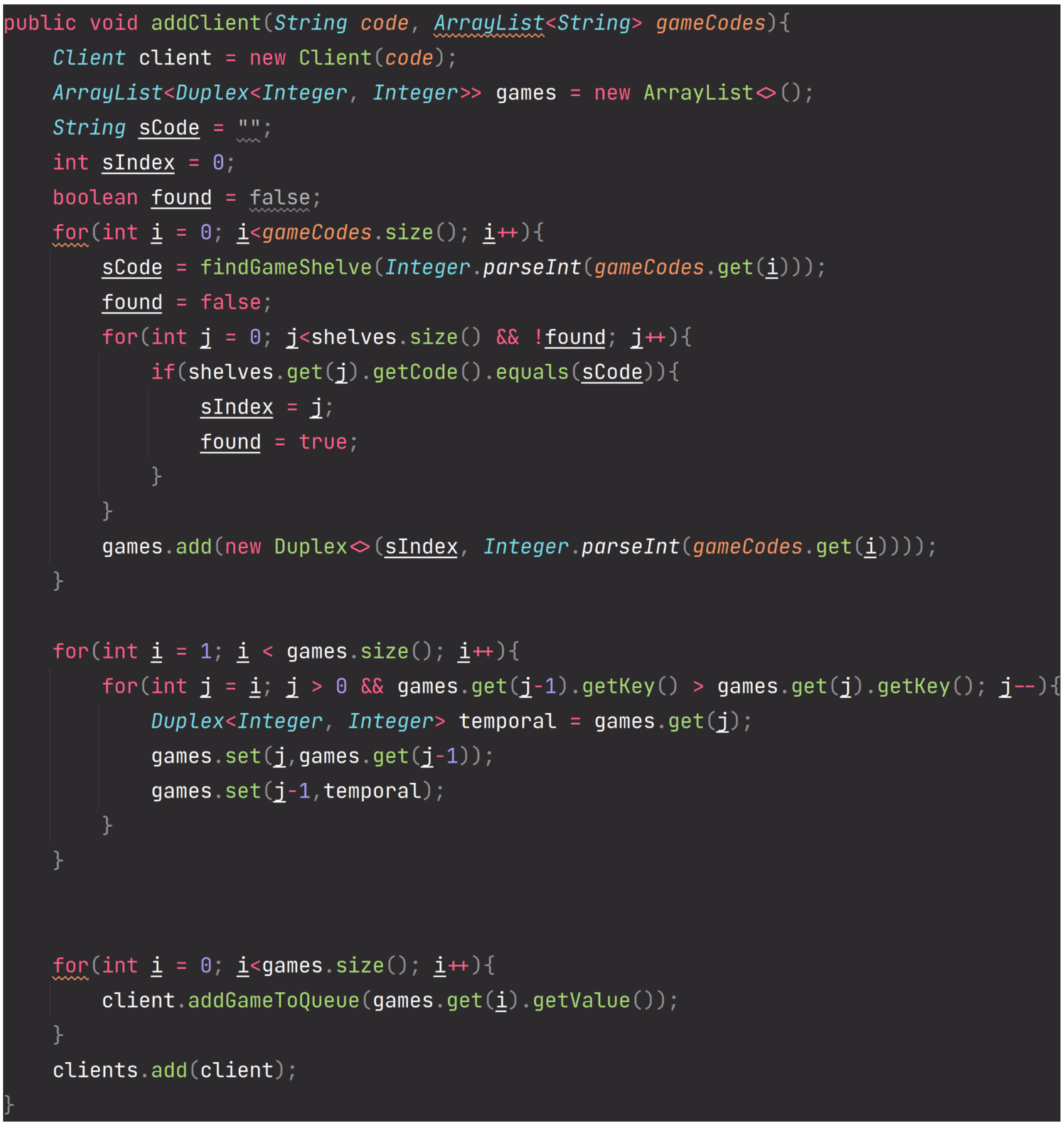
Name:	shelveAddGame
Description:	Add a game to a shelve
Input:	- code: The code of the shelve - gCode:The code of the game that will be added - price: The price of the game that will be added - quantity: The quantity of the game that will be added
Output:	



```java
public void shelveAddGame(String code, int gCode, int price, int quantity){
    Game game = new Game(gCode, price, quantity);
    shelves.get(findShelve(code)).addGame(game);
}
```

C:

Name:	addClient
Description:	Add a client
Input:	- code: The id of the client - gameCodes: A list with the codes of the games that the client wants to buy
Output:	



```java
public void addClient(String code, ArrayList<String> gameCodes){
    Client client = new Client(code);
    ArrayList<Duplex<Integer, Integer>> games = new ArrayList<>();
    String sCode = "";
    int sIndex = 0;
    boolean found = false;
    for(int i = 0; i<gameCodes.size(); i++){
        sCode = findGameShelve(Integer.parseInt(gameCodes.get(i)));
        found = false;
        for(int j = 0; j<shelves.size() && !found; j++){
            if(shelves.get(j).getCode().equals(sCode)){
                sIndex = j;
                found = true;
            }
        }
        games.add(new Duplex<>(sIndex, Integer.parseInt(gameCodes.get(i))));
    }

    for(int i = 1; i < games.size(); i++){
        for(int j = i; j > 0 && games.get(j-1).getKey() > games.get(j).getKey(); j--){
            Duplex<Integer, Integer> temporal = games.get(j);
            games.set(j,games.get(j-1));
            games.set(j-1,temporal);
        }
    }


    for(int i = 0; i<games.size(); i++){
        client.addGameToQueue(games.get(i).getValue());
    }
    clients.add(client);
}
```

D:

Name:	findGameInShelve
Description:	Find a game in the shelves
Input:	- code: The code of the game that wants to find
Output:	- code of the shelve where the game is