Documentation of the Programming Project

Group Members: Imanol Maraña and Borja Moralejo

Date: Monday, 19 October 2020

Subject: Data Structure and Algorithms

Degree: Computer Science

Academic Year: 2nd

School: Facultad de Ingeniería Informática UPV

Table of Contents

Table of Contents **2**

Introduction **3**

First version of the Project **4**

Classes design 4

Description of the data structures used in the project 5

Design and implementation of the methods 7

Second version of the project **8**

Notes of the meetings **8**

Conclusions **8**

Annexes **9**

User guide 9

UML 10

JUnit 10

1. Introduction

In this project we are going to develop a social network from scratch in Java programming language. We will learn in the process of making the different data structures types used in real world applications, their advantages and disadvantages. The main objectives are in a document marked as milestones and we must fulfill the specified points from them.

We have to think and design our data structure for the project, analyzing and calculating the efficiency and measuring the time that takes to finish their tasks, experimentally or by tilde approximation. After that, we will conclude a resolution and then rethink and remake our data structure if we find a better design while studying or after examining our previous designs.

In this document we are reporting our project evolution and development choices, for example our data structures changes or our algorithms functionality. Indirectly we will be reporting our learning process in this subject because we are going to explain our different versions of the project.

The JUnit test cases and UML diagrams are a helping hand explaining our work and the integrated user documentation or guide will help the user know the basic functionality.

2. First version of the Project

Our first version only implements the first milestone of the description of the Programming Project and some extra sections. How it implements those parts are explained in the following paragraphs, first we will specify what our program currently does.

The first milestone is entirely implemented: It has an initial menu that you can use for interacting with the social network. It can load people and their relationships into the social network from a file inside res folder and print the people inside the network into a file.
The other implemented sections are: You can search people by their attributes and get the other people that have the same attribute. Also, you can remove a person from the network and clean memory.

2.1. Classes design

This is a brief description of the classes used in this project. Each class has it's full documentation in /doc.

The main classes are: `SocialNetwork` and `DataHolder`

`SocialNetwork` class has the main method and the menu on it.

`DataHolder` class as the name indicates, contains the data of the social network.

`Person` class has the attributes inside and it is the object that has to be stored, as well as its attributes.

Inside the package `adt` we have all the abstract data types that we are using in the project.

Inside `abstractDataTypesImplemented` we have all the classes that are implementing the ADTs.

Inside `exceptions` package we have… exceptions classes.

2.2. Description of the data structures used in the project

We have two main types. Data structures for the attributes and data structures for the relationships. But first we are going to explain how Person class has their attributes stored.

Person class uses two data structures that are going to be explained further down, `DataBlockADT<String, String>` and `NodeADT<String>`. Person has an array of arrays of DataBlockADT for the attributes and a NodeADT for the relationships.

We have chosen this approach because if the user or person decides that it doesn't want to put that attribute, their array length for that attribute is 0 and we know that the user didn't enter any data.

For the attributes:

For the main data storage system we have the following structure:

All data is stored inside DataBlocks objects, they work as a node but they have another functionality. In our case we use a `DataBlock<String, String>`, the class has a key type and a collection element type. The DataBlock holds all elements that have the share key. For example, if we have a datablock holding the "*Donostia*" key String of the attribute home and we make an algorithm to add elements that have the same attribute, in this case, people's id that have a home in Donostia.

With this element we have all the people that are related by an attribute and if we need to search somebody related, we can just use the datablock for getting the ids' that are related. For this reason we need an algorithm and a data structure that can access efficiently to the asked datablock.

Because of that we have the following structure in DataHolder:
It contains two main variables *personHashMap* `HashMapADT<Person`(value)`, String`(key)`>` and *stringHashMaps* if of the type `HashMapADT<DataBlock<String, String>`(value)`, String`(key)`>[]` with a length of the number of attributes minus the id because Person class has a unique id with their own getter.

With that type of structure we can have each datablock which stores a unique attribute value and all the related ids hashed by the value that stores. So, if we search for the people that were born in Donostia, we ask the dataholder inside the *stringHashMap* hashmap array in the index assigned for the attribute *birthplace* for the datablock with the key "Donostia". Supposing that the hashmap has the asked datablock, it will return it and we can then proceed to use the second main variable, *personHashMap*.

The variable *personHashMap* is a hashmap of Person with their id (String) as keys. With this hashmap we can search efficiently the Person by their id.

To sum it up, this data structure allows a fast and efficient method for searching related people for the cost of some space, but nowadays it's easier to expand the capacity than the speed of the processor.

For the relationships:

At the moment of loading a person into the network, a node is created and assigned to that person with that person's ID as it's only content. Therefore, we now have a node assigned to each of the persons loaded, ready to operate with them.

When we add a new friend relationship between two people, the node of each other stores the other person's ID in an ArrayList, using an alphabetical order, so a “link” is made between the two of them. Whenever we want to unlink them, we just need to delete each other's ID from the friend list. Also, there's a counter that indicates the number of links or the number of friends that someone has. If we make a link, that number increases, and if we unlink them, it decreases.

If we want to operate with the friends of anyone stored in our database, as long as the desired people are linked to each other, we can get their node, as well as their information, without any problem. To make this process faster, we previously ordered the friends ID alphabetically, so we just need to make a quick search and we would know if both of them are linked or not. After that, accessing the friends info is as easy as knowing whether they are linked or not, we need to find the link to return it after all.

This way, we can operate with the relationships of people with ease and quickness, as we have an efficient way of storing and searching for info.

2.3. Design and implementation of the methods

We are mentioning and giving a brief description of the method, for more information, please go into the javadoc documents.

In Person.java

Constructor `Person(String combinedData)`:
It decomposes the string and gets the attributes that are separated in a .csv format.

Function `getAtributesRelatedDataBlocks(int attribute)`:
Returns an array with the Datablocks with the given attribute, for example, all their homes.

In DataHolder.java

Function `constructorUseDataBlock(int attribute, String key)`:
This method is used to add the Person into the network, for example we can create a Person object individually but we need to get or create the attribute's datablock for correct network functionality.

Function `getDataBlockOf(int attribute, String key)`:
Is used for getting related people by the given attribute.

Method `addPersonToNetwork(Person p)`:
Links their attributes with the network's datablocks. It uses the constructorUseDataBlock function.

Method `loadFile(String filename, int option)`:
Tries to load the file with the given name and loads people into the network or loads their relationships based on the option parameter.

Function `getPersonByID(String id)`:
Gets the person with the given id from the hashmap, if there's no person with such id, it throws an exception.

Function `searchPeopleByAttribute(int attribute,  String value)`:
Gets the people that have the given attribute.

Method `printIntoFile(String fileName)`:
Prints all the people from the network into the filename.

3. Second version of the project

4. Notes of the meetings

First meeting:

We discussed the data structure that our project was going to have, we decided that it needed a fast accessible data structure that doesn't need to do many comparisons, so we thought about an indexed structure, and from there we found out that hashmaps work similar to that.

Second meeting:
We divided the work we had to do for this project. Imanol would be working on relationships and the menu and Borja on the Person and the data storage structure or system.

Third meeting:
We shared our advancements and our problems, we sorted out the majority of them.

Fourth meeting:
We explained to each other how exactly our part of the project worked.

5. Conclusions

This project is unfinished but we have learned that there are more data structures than arrays and arraylists and their uses are endless.

6. Annexes

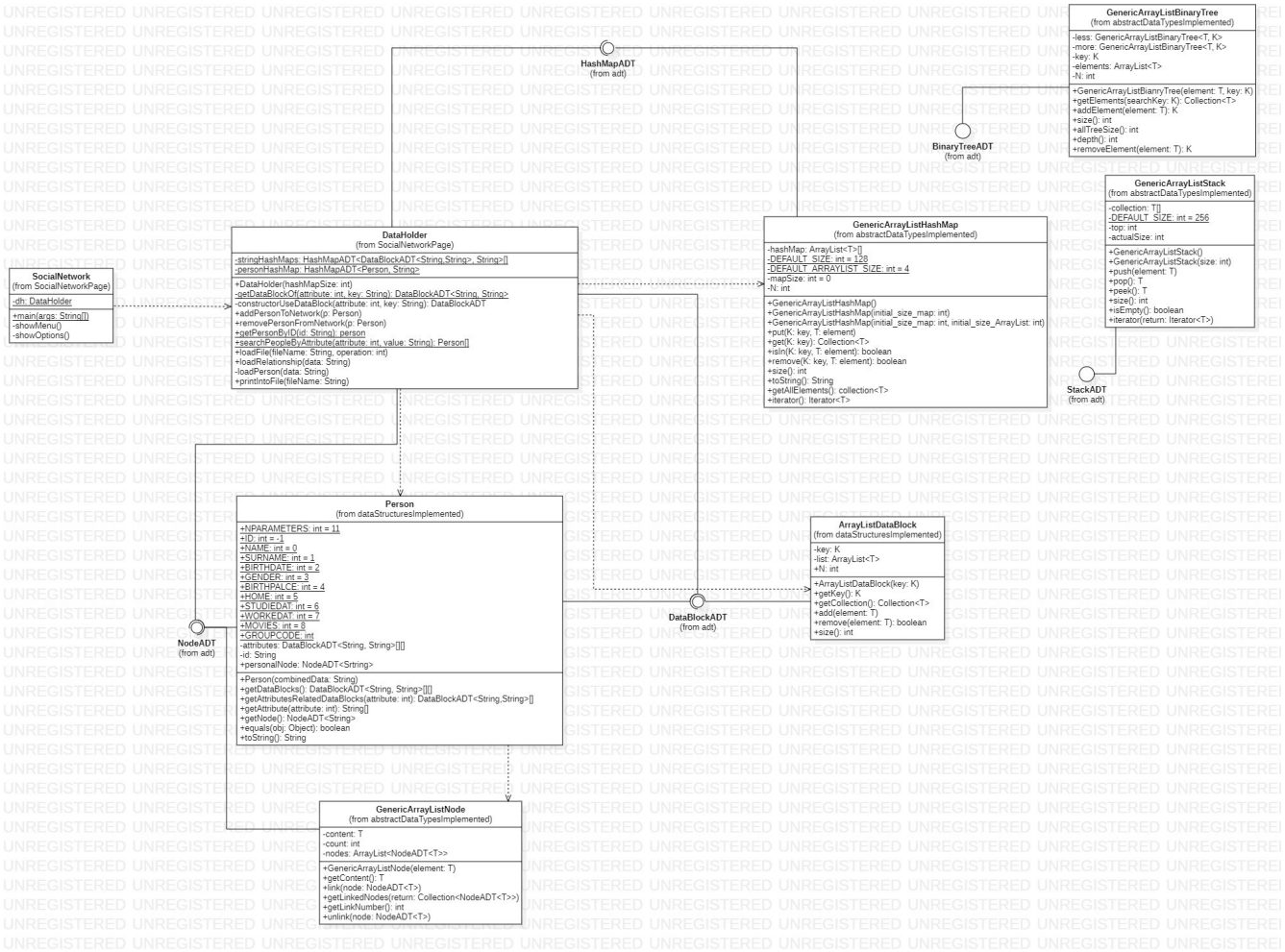
6.1. User guide

Menu Options:

1. Load file: Loads a text file located in the res directory given file name. That file must not contain information with more than 11 attributes defined by “,”. Then, the information within the file is used to create a person, which will be added to the social network.
2. Load relationships: Loads a text file located in the res directory given file name. That file must not contain information with more than 2 attributes defined by “,”. Then, the information within the file is used to create a friend relationship between 2 people, which will be added to the social network.
3. Print into file: Gets the information of all the people added into the network and prints it into a file located in the res directory with a given name. It doesn't print any relationship.
4. Search people: Given a number associated with an attribute and a value, searches for any person with the given value in the given attribute and prints its information. If the attribute is not ID, it may print more than one person.
5. Add people: Given the information of a person in a correct order, creates and adds that person into the social network. That information must not contain more than 11 attributes, but must have at least 1 attributes: the ID. If adding more than one information to a single attribute is desired, put a “;” between the information.
6. Remove person: Given the ID, deletes a person with that ID from the network if exists.

6.2. UML

Inside the project folder we have the class diagrams. Here is a picture.



We recommend opening the .mdj file for a detailed view.

6.3. JUnit

We have used the library JUnit for the test plan of our project. It tests the loading, printing and storing methods of the DataHolder. It tests the datablock's methods and attribute obtention. Said test has a coverage of 70% of the project, the uncovered classes are the menu class and unused implemented ADTs.

Search in the project for more information.