Programming project

Date: Sep/09/2021 Group Members:

Pablo Jurado

Borja Gómez

Erik Cembreros

Subject: Data Structures and Algorithms Degree: Informatics Engineering Academic Year: 2022/2023

School: Facultad de Informática

Table of Contents

Abstract 3

1. Introduction 3

2. UML of the Project 3

3. First version of the Project 4

    3.1. Classes design 4

    3.2. Description of the data structures used in the project 4

    3.3 Design and implementation of the methods 4

4. Second version of the Project 5

    4.1. Classes design 5

    4.2. Description of the data structures used in the project 5

    4.3. Design and implementation of the methods 6

5. Final version of the Project 7

    5.1. Classes design 7

    5.2. Description of the data structures used in the project 7

5.3. New implementations 7

    5.4. Design and implementation of the methods 8

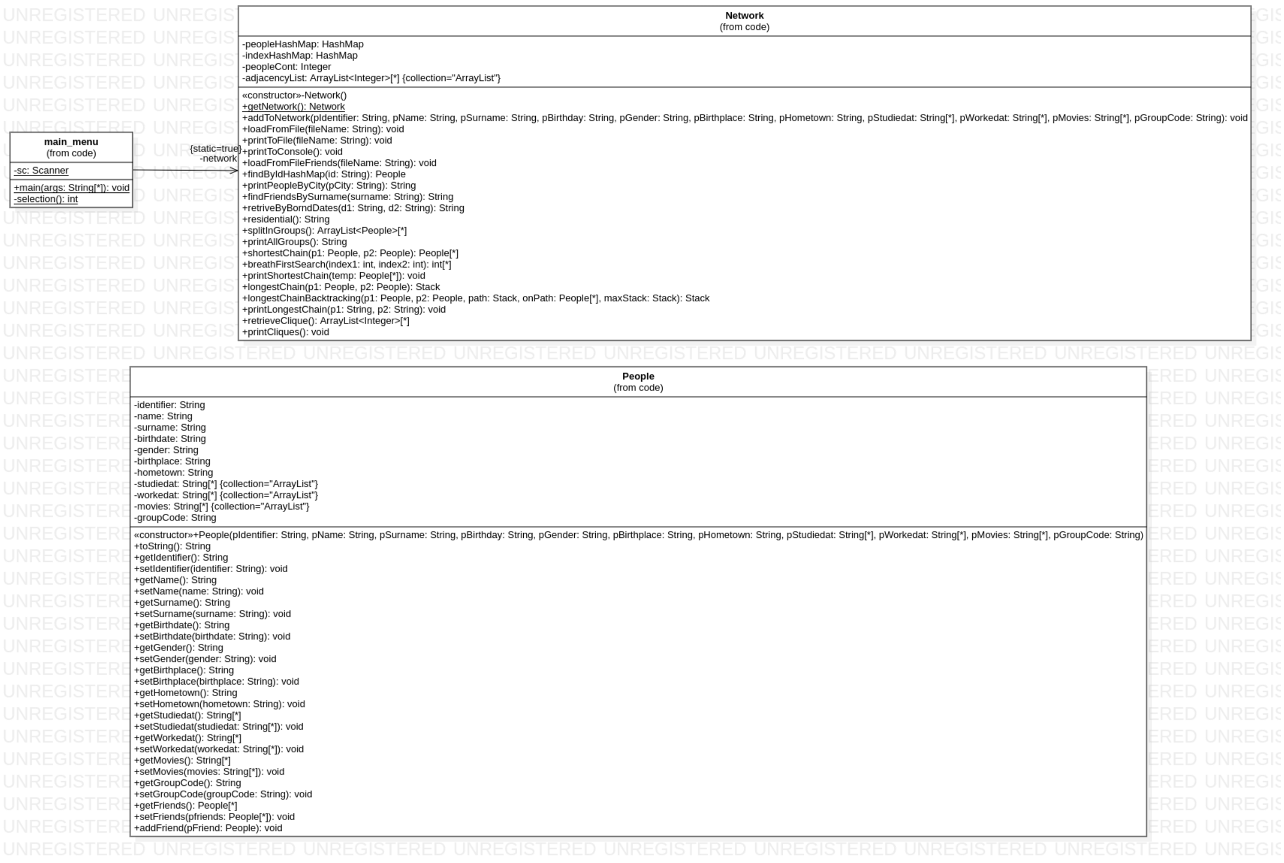
Abstract

In this project we have worked on a program that deals with a software application of a social network. The social network is made up of people that may be linked among each other if there is a friendship relationship among them. To start with the work, the first thing we did was to create the program we were going to work on and the text documents "people.txt" and "friends.txt" in order to have the content to work with in the program. The file “people.txt” describes the collection of people in the network. The people of this file are considered to belong to the social network and the file “friends.txt” describes the friendship relation among people as pairs of identifiers.

1. Introduction

As it is commented in the abstract the project consists mainly of a program with a menu, in which there are different tasks to work with, in which the text files "people.txt" and "friends.txt" are processed, such as define a list of identifiers or to create a file linking those identifiers in pairs, whilst assuring that they form a clique.

2. UML of the Project


Network
(from code)
-peopleHashMap: HashMap
-indexHashMap: HashMap
-peopleCont: Integer
-adjacencyList: ArrayList<Integer>[*] {collection="ArrayList"}
«constructor»-Network()
+getNetwork(): Network
+addToNetwork(pIdentifier: String, pName: String, pSurname: String, pBirthday: String, pGender: String, pBirthplace: String, pHometown: String, pStudiedat: String[*], pWorkedat: String[*], pMovies: String[*], pGroupCode: String): void
+loadFromFile(fileName: String): void
+printToFile(fileName: String): void
+printToConsole(): void
+loadFromFileFriends(fileName: String): void
+findByIdHashMap(id: String): People
+printPeopleByCity(pCity: String): String
+findFriendsBySurname(surname: String): String
+retriveByBorndDates(d1: String, d2: String): String
+residential(): String
+splitInGroups(): ArrayList<People>[*]
+printAllGroups(): String
+shortestChain(p1: People, p2: People): People[*]
+breathFirstSearch(index1: int, index2: int): int[*]
+printShortestChain(temp: People[*]): void
+longestChain(p1: People, p2: People): Stack
+longestChainBacktracking(p1: People, p2: People, path: Stack, onPath: People[*], maxStack: Stack): Stack
+printLongestChain(p1: String, p2: String): void
+retrieveClique(): ArrayList<Integer>[*]
+printCliques(): void
main_menu
(from code)
-sc: Scanner
+main(args: String[*]): void
-selection(): int
{static=true}
-network
People
(from code)
-identifier: String
-name: String
-surname: String
-birthdate: String
-gender: String
-birthplace: String
-hometown: String
-studiedat: String[*] {collection="ArrayList"}
-workedat: String[*] {collection="ArrayList"}
-movies: String[*] {collection="ArrayList"}
-groupCode: String
«constructor»+People(pIdentifier: String, pName: String, pSurname: String, pBirthday: String, pGender: String, pBirthplace: String, pHometown: String, pStudiedat: String[*], pWorkedat: String[*], pMovies: String[*], pGroupCode: String)
+toString(): String
+getIdentifier(): String
+setIdentifier(identifier: String): void
+getName(): String
+setName(name: String): void
+getSurname(): String
+setSurname(surname: String): void
+getBirthdate(): String
+setBirthdate(birthdate: String): void
+getGender(): String
+setGender(gender: String): void
+getBirthplace(): String
+setBirthplace(birthplace: String): void
+getHometown(): String
+setHometown(hometown: String): void
+getStudiedat(): String[*]
+setStudiedat(studiedat: String[*]): void
+getWorkedat(): String[*]
+setWorkedat(workedat: String[*]): void
+getMovies(): String[*]
+setMovies(movies: String[*]): void
+getGroupCode(): String
+setGroupCode(groupCode: String): void
+getFriends(): People[*]
+setFriends(pfriends: People[*]): void
+addFriend(pFriend: People): void

3. First version of the Project

3.1. Classes design

The total work is made up of three different classes: main_menu, network and people. In "main_menu", as the name suggests, is the menu with which the program is told which task to perform. Each of the tasks written in this class use the methods of the other classes. The first task define a list of identifiers using “people.txt”. Then the second task load the “relationships” into the network and the third task prints out the people of the network. Then if you want to finish working with the program it is necessary to write 0. Then in the class “network” we have define all the methods that are used in the class “main_menu” to work with the implemented menu. There you can find methods such as, “loadFromFile” or “printToFile”. And finally, in the class “people” we have implemented attributes and the constructor of the object people and accompanied of all its getters and setters. Also, in the class “people” there are define the methods “addFriend” and “toString”.

3.2. Description of the data structures used in the project

Principally we have worked with the ArrayList People in which the object people are stored. On the other hand, inside the object People there are different attributes that are also ArrayList: “studiedat”, “workedat” and “movies”.

3.3 Design and implementation of the methods

1. Method **`loadFromFile()`**

```
Method that loads from the file all the people given as example.

@param fileName name of the file we want to do the import from.
```

2. Method **`printToFile()`**

```
Method that prints on a file the people.

@param fileName name of the file where we want to print the people.
```

3. Method **`finfById()`**

```
Method that search a person by id in the hashmap.

@param id Identifier of the people we want to find.

@return person with that id.
```

4. Method **`LoadFromFileFriends()`**

```
Method that loads from a file all the people given as example.
@param fileName name of the file we want to do the import from.
```

4. Second version of the Project

4.1. Classes design

As in the first part of the work, the program is made up of the same three classes as in the previous part, since the only thing we are going to do is to update and add new functions to our social network. In this case, the new methods we have created are as follows. To start with, a method that, given a city, writes which people were born there. Also, the option of being able to print the friends that a person has by means of the name and surname of that person. On the other hand, by typing two dates you will also be able to see the date of birth, name and surname of those people who were born between those two dates. Another option available on the social network, is that given a set of identifiers in a file named "*residential.txt*", recover the values of the attributes name, surname, birthplace and studied at of the people on the network whose birthplace matches the hometown of the people who are described in "*residential.txt*". And finally, the last method created for the social network is one in which you group all the people in different groups that have the same collection of movies, which will be grouped in an arrayList.

4.2. Description of the data structures used in the project

As in the first part we have used the same structures as we have to continue using the same objects and arryLists to make work the social network. But in this case, we have implemented a new list of examples in the method residential in which there is a list of identifiers. Also in the last method we have created a new arrayList to make groups of people that have the same collection of movies.

4.3. Design and implementation of the methods

1. Method **printPeopleByCity()**

Method that returns a string with the people from the hometown given by parameter pCity

@**param** pCity name of the city

@**return** string with the result of the method

2. Method **findFriendsBySurname()**

Method that returns a string with the friends of a person by his/her surname

@**param** surname surname of the person

@**return** string with the result of the method

3. Method **retriveByBornDates()**

Method that returns a string with people born between two dates, and sorts them by born date, surname and name

@**param** d1 first date

@**param** d2 second date (higher than d1)

@**return** string with the result of the method

4. Method **residential()**

Method that returns a string with the name, surname, birthdate and place of study of the people whose birthplace is the same hometown of the different people in residential.txt

@**return** string with the result of the method

5. Method **splitInGroups()**

Method that splits all the people in the network into groups with the same films

@**return** arraylist of groups(arraylist of people)

5. Final version of the Project

5.1. Classes design

As in the previous two parts, the work is still made up of the 3 classes we have been using previously, as the purpose of this work is to continue updating the network class. First, we are asked to create the shortest chain linking two people by the friends they have. To do this we have created three different variables, one to create the chain of people, one to create the chain of friends, and one to create the chain of friends, other that do the same but using breath first search and the method which prints the chain of people. On the other hand, we are asked to create again chains of people with the same characteristics as in the method of creating the shortest chain, but this time instead of being the shortest chain, it has to be the longest chain of people, then the worker method that instead of returning an arraylist returns a stack with the longest chain between two people and the method that prints the string. And lastly, we are asked to create a method that returns a list of all the cliques of more than 4 people in the network. To know what the method does, we must understand that, a clique is a group of friends in which each person has friendship with each other. And finally, to complete the last task we will create a method that will print out all then cliques of more than 4 people in the network.

5.2. Description of the data structures used in the project

At the start of this version of the project we have relished that we could use hashmaps for a better structure of the network. We have created two variables of hashmap, one that given the index of the person gives us the object people, and another one that given the object people gives us the index of the person. Doing this change we have had to change some implementations of some methods so that they are compatible with hasmaps. Although we have done use of hashmaps for structuring the network, in most methods we have done use of arraylists for developing it.

5.3. New implementations

To improve the try and catch of errors, we have developed some personalized exceptions in order to return to the user a message that specifies the error committed by the user.

These are the exceptions added to the project:

PersonAtNetwork extends **RuntimeException**

FriendsAtNetwork extends **RuntimeException**

PersonNotFound extends **RuntimeException**

RelationNotExist extends **RuntimeException**

5.4. Design and implementation of the methods

1. Method **shortestChain()**

This method returns a LinkedList of People with the shortest chain between two people

@param p1 the first person

@param p2 the second person

@return a linked list with the shortest chain between the two people

2. Method **breathFirstSearch()**

This method calculates the first breath search, that we use to calculate the previous method.

Method that calculates a chain between two people using breath first search

@param index1 the first person

@param index2 the second person

@return an arraylist with the chain between the two people

3. Method **printShortestChain()**

This method prints the LinkedList returned by the method shortestChain()

@param p1 first person for the worker

@param p2 second person for the worker

@param path stack with the current path

@param onPath set with the people that are on the path

@param maxStack the max stack with the longest chain

@return a stack with the longest Chain between two persons

4. Method **longestChain()**

Obtains the longest chain of relations between person1 and person2 users in the Social Network.

@param p1 Initial Person's identifier.

@param p2 Final Person's identifier.

@return ArrayList of Persons with the longest chain of relations.

5. Method **longestChainBacktracking()**

Worker method for longestChain that returns a stack with the longest Chain between two persons

Because of the nature of the problem, the method uses backtracking and checks all the links. And because the network has a high amount of people it will take some time to look at all the edges in the network for a link.

@**param** p1 first person for the worker

@**param** p2 second person for the worker

@**param** path stack with the current path

@**param** onPath set with the people that are on the path

@**param** maxStack the max stack with the longest chain

@**return** a stack with the longest Chain between two persons

6. Method **printLongestChain()**

Prints the longest chain between two persons

@**param** p1 first person

@**param** p2 second person

7. Method **retrieveClique()**

Method that returns a list of all the cliques of more than 4 people in the network

@return an arraylist with all the cliques

8. Method **printCliques()**

Method that prints all the cliques of more than 4 people in the network