DOKUZ EYLÜL UNIVERSITY
ENGINEERING FACULTY
DEPARTMENT OF COMPUTER ENGINEERING

CME 2210
Object Oriented Analysis and Design

CEMETREE

by
Boran Bereketli – 2022510105
Bedirhan Yenilmez – 2022510159
Alperen Dönmez – 2022510113
Ramazan Denli - 2022510111

IZMIR
20.05.2024

CHAPTER ONE

INTRODUCTION

Cemeteries have a problem of being unorganized and hard to navigate. Some graves are even unknown. This project aims to solve this by providing a way to store and manage people's and cemetery's information. This is a cemetery management and navigation system designed to help finding where people are buried to and get information about them.

The system is designed to store cemetery information and buried people's information and assist finding and choosing a cemetery. It will be able to connect people, find where their relatives are buried to and indicate their location.

The information about buried people (name, surname, birthdate, death date, relatives, death cause, cemetery information etc.) is stored in a data structure and can be saved and loaded from a file. The records of visitors(id, name, surname, date etc.) can be kept by the system.

The people can be filtered and sorted by their name, surname, age, cause of death. Some statistics such as top death causes, average age of dead people, male/female ratio, cemetery distribution can be provided according to the user's requests.

This software can be utilized by city managements or countries.

CHAPTER TWO

REQUIREMENTS

2.1 External Interfaces

- Users and administrators can interact with the system.
- Text files are used to read/write data.

2.2 Functions

- Maintain records of cemetery locations, names, and other relevant details.
- Maintain detailed records of people including name, surname, birthdate, death date, cause of death, relatives and cemetery information.
- Track and store visitor information including visitor ID, name, surname, and visit dates.
- Find a deceased person’s location within a cemetery. Include an option to search for alive people as well.
- Locate relatives.
- Locate specific cemeteries and provide details about them.
- Add, remove and edit people or cemeteries from the command line.
- Save and Load Data: Save the current state of the system to files and load it back.
- Update the status of individuals as dead or alive.
- Assign or revoke administrative rights to/from users.

- Filter and sort records by name, surname, age, cause of death, and other criteria.
- Generate statistics such as top causes of death, average age of deceased individuals, gender ratio, and cemetery distribution.
- Show statistics for all selected cemeteries collectively.
- Display detailed information about a specific person or cemetery.
- Log a visit to a deceased person’s grave.
- Get Visitor List: Retrieve the list of visitors for a specific person.

2.3 Performance Requirements

- Relatives are expected to be found in less than 1 second.
- Visitor list search must take less than 1 second.
- The program is expected to search for people in less than 10 seconds.

2.4 Logical Database/File System Requirements

- The program facilitates reading and writing operations from text files.

2.5 Design Constraints

- Cemeteries are designed to include 20000 people at most.
- Only administrators can add or remove people.
- The program needs 3 text files to work.

2.6 Software System Quality Attributes

- HashMap and Graph data structures are used to lower wait times.
- Software must be reliable, maintainable, efficient, and testable.
- Software mustn’t crash under any circumstances and give error messages.

2.7 Object Oriented Models

- 2.7.1 Analysis Class Model (Static Model)

- Cemetree class is used to store all people.
- Cemetery class stores all information about the cemetery.
- Person class is used to represent a person.
- Address class stores an address.
- Date class stores a date.

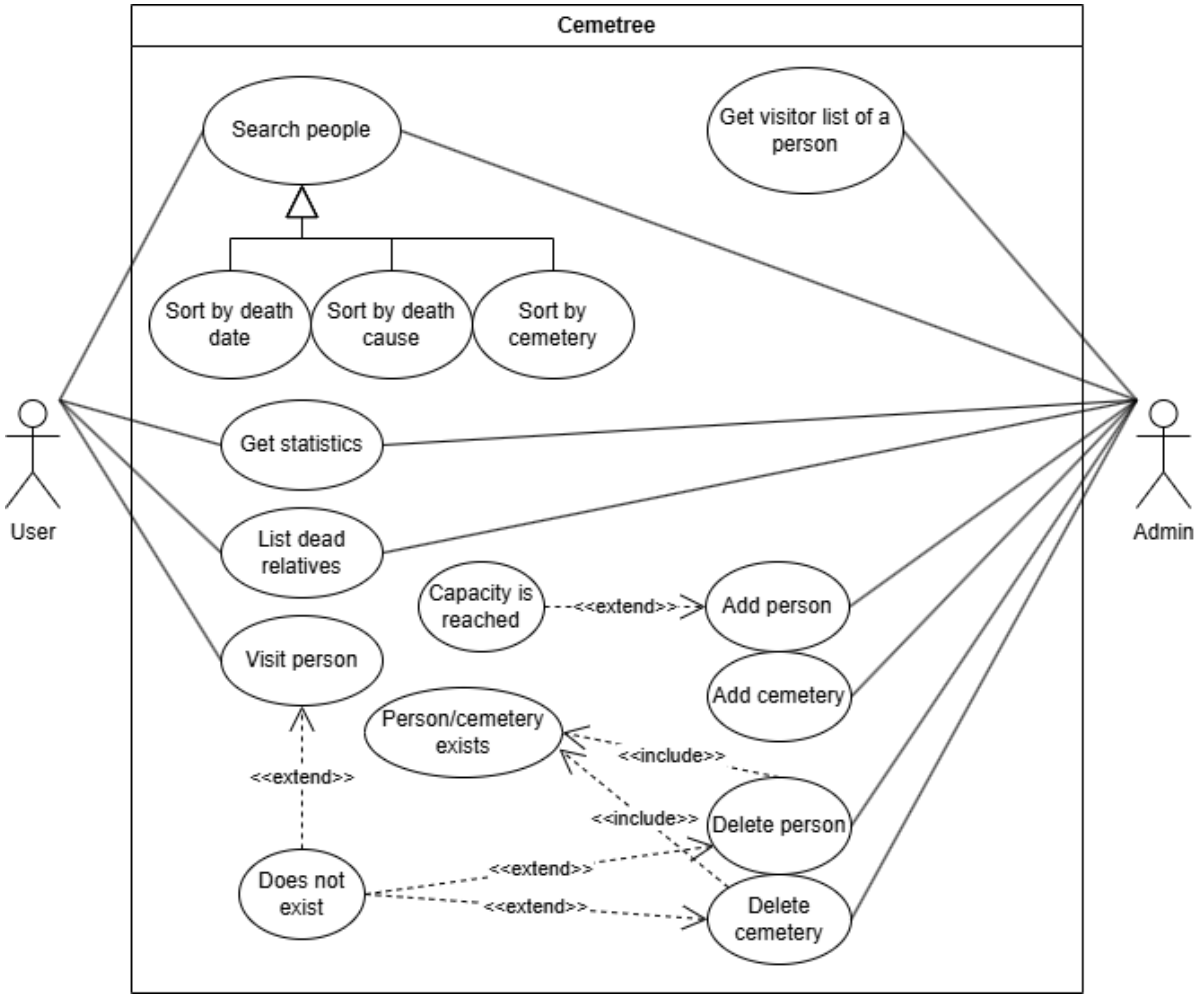
- 2.7.2 Analysis Collaborations (Dynamic Model)

- Cemetree is the main class and stores all the cemeteries and all people with connections.
- All people’s information is manipulated from the Cemetree class.

CHAPTER THREE

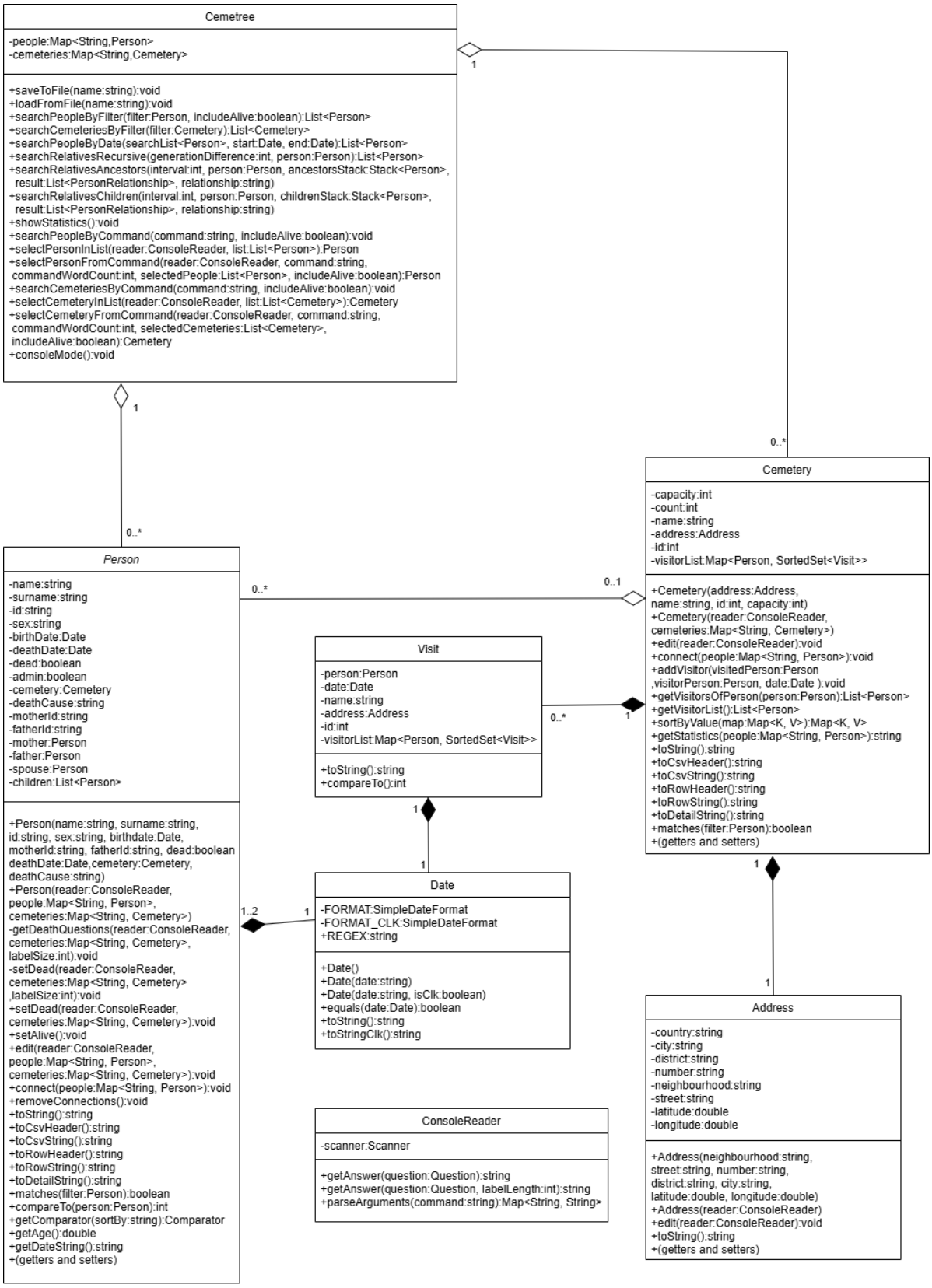
UML DIAGRAMS

USE CASE DIAGRAM



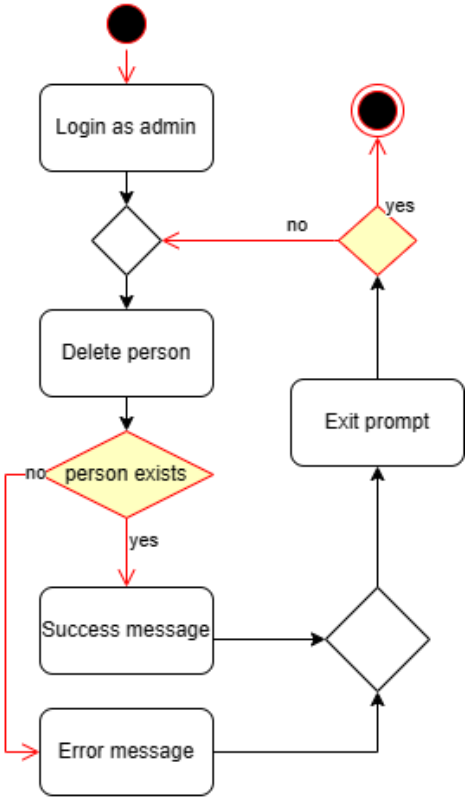
This diagram depicts the general operations the user and admin actors can use. While the user can list relatives, view statistics, search people; the admin can additionally view the visitor records, add people, add cemeteries, delete people and delete cemeteries. Various limits and conditions are also shown like the cemetery capacity and the existence of the person or cemeteries.

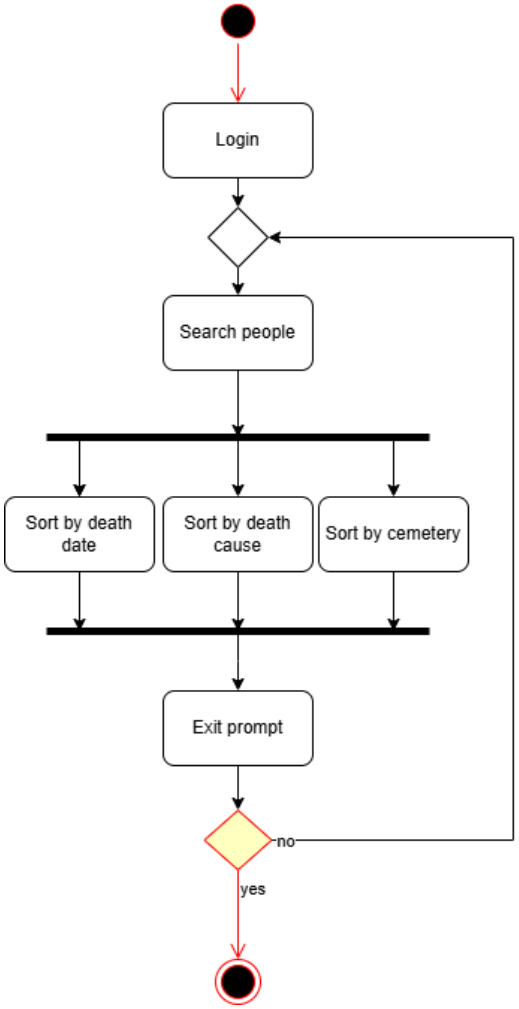
CLASS DIAGRAM

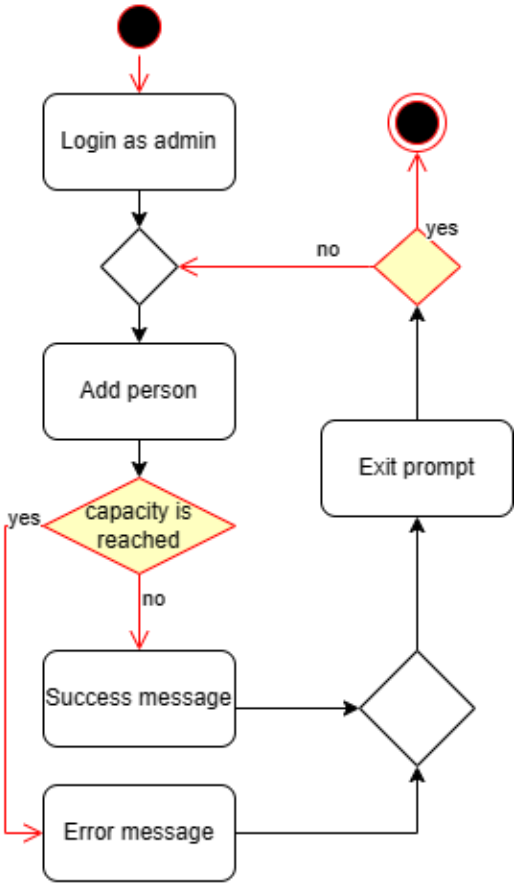


The class structure of the program is shown in the diagram above. The *Cemetree* class contains all people and cemeteries. Zero or more cemeteries and people can be contained in the class. It is the class where all objects are managed from. The other classes hold the data of their respective types. *Person* and *Cemetery* classes hold person and cemetery data. Each cemetery has exactly one address. *ConsoleReader* class is a utility class that helps in console reading operations.

ACTIVITY DIAGRAM







Each activity needs login as the first action.

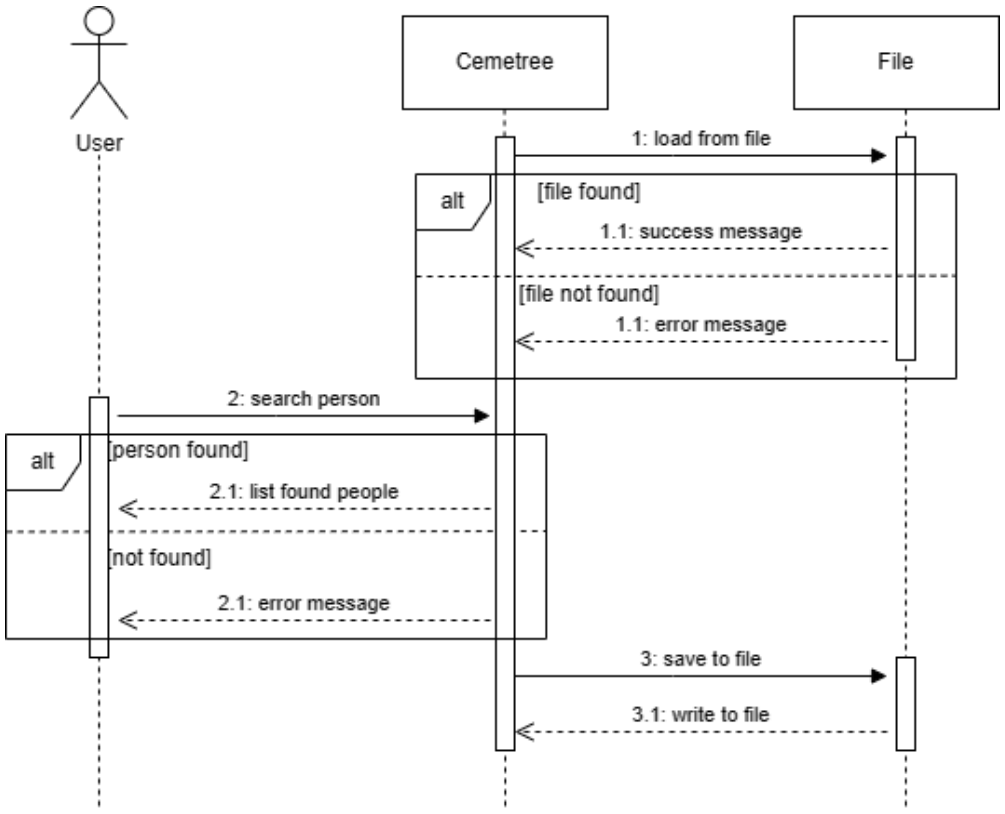
The deletion of people checks the existence of the person, deletes the person and shows a message to the user.

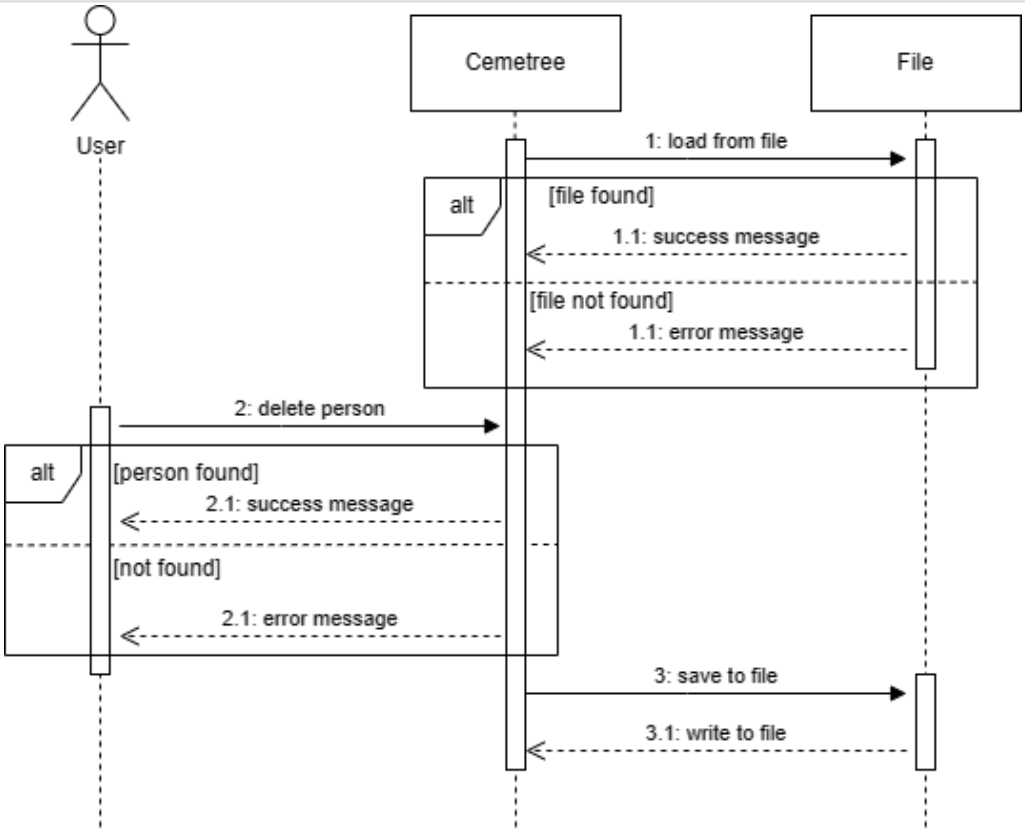
The addition of people checks the cemetery capacity, adds the person and shows a message to the user.

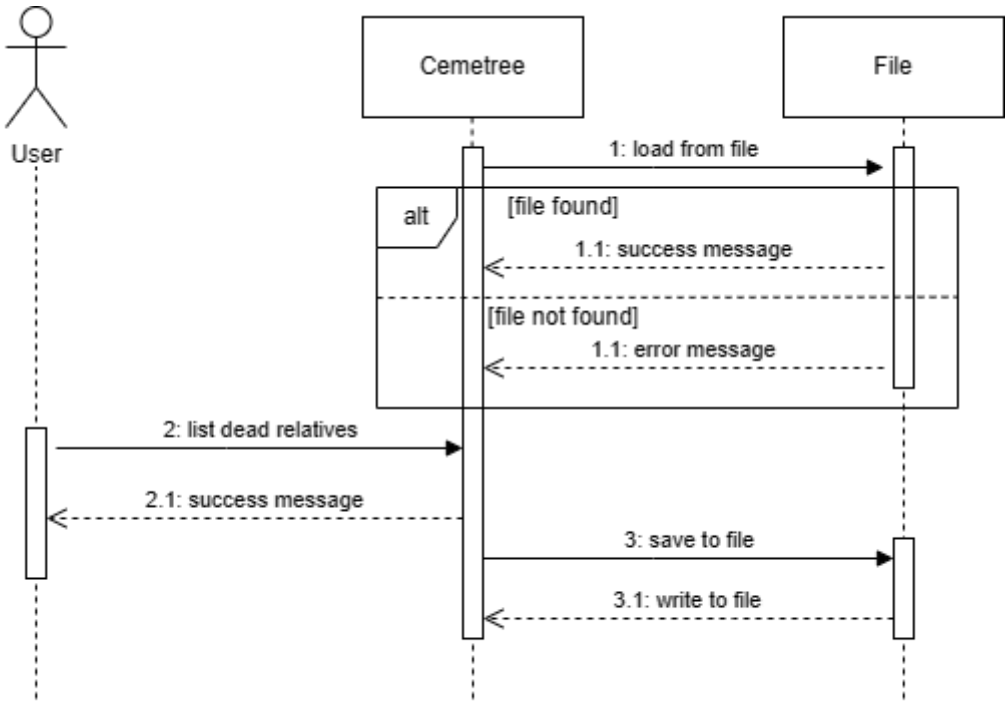
The results of the search function can be sorted by death date, death cause or cemetery.

An exit prompt is shown after all actions.

SEQUENCE DIAGRAM







Each diagram has loading from files as the first action. It returns a message after the files are read.

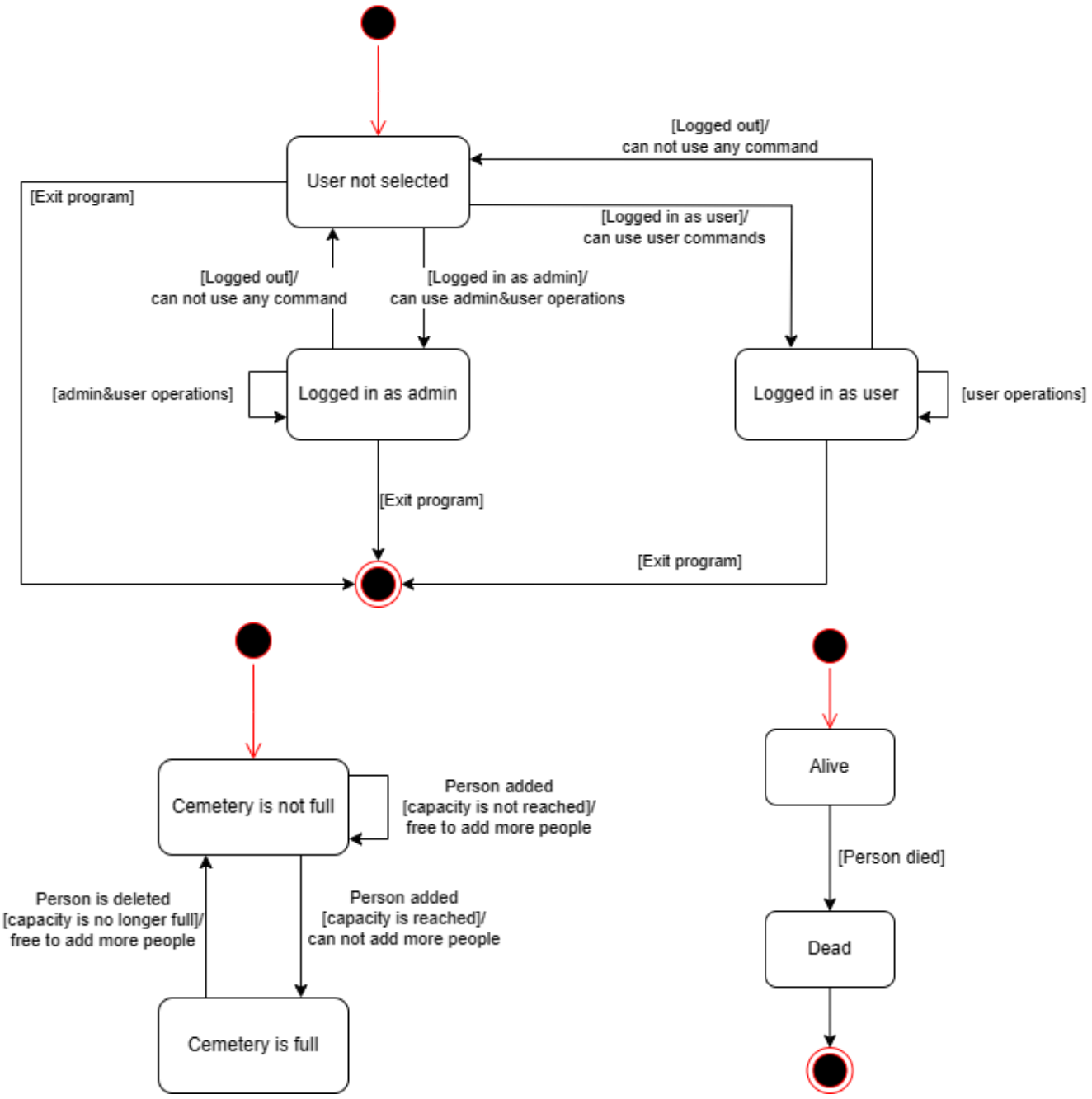
The first diagram shows the search operation. It shows the list of the matched people.

The second shows the delete operation. It checks if the person exists and returns a message according to the result.

The last diagram depicts the list dead relatives' function.

All diagrams save the changes to files and show a message according to the result as the last action.

STATE DIAGRAM



The first diagram shows the login states. The user can login either as user or as admin. User and admin have their respective operations they can use. They can exit the program or log out at any time.

The second diagram shows the cemetery capacity states. It is either full or not full. People can be added when it is not full.

The last diagram shows the state of the person.

CHAPTER FOUR

IMPLEMENTATION

The *Cemetree* class houses all parts of the system. It has functions to save to file, load from file, search people or cemeteries by using a filter and sort them, search people's dead relatives, show statistics about all cemeteries, filtered cemeteries or a specific cemetery.

The program needs 3 files to store people, cemeteries and visitor lists. The program loads the files, calls the *consoleMode* function and saves to files after exiting.

ConsoleReader class helps to get input from the user or parse commands.

Date class is inherited from the *GregorianCalendar* class and date formats are added to it. It also has specialized functions to parse date from string and equals function just for the day, month and year.

Person, *Cemetery* and *Address* classes have special constructors to take input from the user with using *ConsoleReader*. They also have *toDetailString* functions that generates a string showing the details of them.

The *consoleMode* function in the *Cemetree* class provides 21 unique commands to manage the system. The *help* command's output is shown in the page below.

Iterator design pattern is used in some parts of the code.

The rules for the usage of the commands are shown below.

- None of the commands, arguments or flags are case sensitive.
- The command, arguments and flags are seperated by spaces.
- Flags can have spaces.
- Argument keys can not have spaces.
- Argument values need quotation marks if they include spaces.
- An asterisk can be used to use no filters.

Here are some examples:

> *COMMAND ARGUMENT_A=<a> ARGUMENT_B=<b> FLAG*

> *ADD PERSON*

> *SEARCH PERSON NAME=john BIRTH_DATE=12/02/2008 INCLUDE ALIVE*

> *SEARCH CEMETERY CITY=İstanbul NEIGHBORHOOD="Kemer Mahallesi"*

> *SHOW STATISTICS*

> *SHOW STATISTICS ID=34-046*

> *SHOW STATISTICS CITY=İstanbul GROUP*

> *SHOW STATISTICS * GROUP*

> *SEARCH RELATIVES INTERVAL=3*

> *HELP VISIT PERSON*

> These conditions are checked inside the *consoleMode* function for each command by using regex expressions. The admin status of the user is checked too for restricted commands.

Use help <command> for more information on a specific command.

ADD PERSON	Adds a new person
REMOVE PERSON	Removes a person
EDIT PERSON	Edits a person
SET DEAD	Changes a person's status to dead
SET ALIVE	Changes a person's status to alive
SET ADMIN	Changes a person's admin status
SET USER	Changes a person's admin status
SEARCH PERSON	Searches for a person (Use "include alive" flag to include alive people at search)
SEARCH RELATIVES	Searches for relatives
ADD CEMETERY	Adds a new cemetery
REMOVE CEMETERY	Removes a cemetery
EDIT CEMETERY	Edits a cemetery
SEARCH CEMETERY	Searches for a cemetery
SHOW STATISTICS	Shows statistics (Use "group" flag to show statistics of all selected cemeteries)
VIEW	Views details of a person or cemetery (Use "view person" or "view cemetery" to view corresponding details)
VISIT PERSON	Visits a person
GET VISITOR LIST	Gets the visitor list of a person
HELP	Shows help for <command>
LOGOUT	Logs out
CANCEL	Cancels current operation
EXIT	Exits the program

CHAPTER FIVE

CONCLUSION AND FUTURE WORKS

In this chapter, a summary and future works must be written.