Business management

> Showcase project built from scratch to demonstrate OOP features for Object Oriented Programming subject in Politechnika Krakowska.

Table of Contents

General Information

C++ program designed to showcase the functionality of OOP. It represents a simple business in which there are departments and employees. It has a shell-like command line interface through which the user is able to manage objects and apply business logic to them. It mocks a database by using text files and admits configuration for both database files and help files.

Functioning

The program is capable of storing and managing employees and the departments in which they work. For each employee, it stores information about it's name, salary, and department. For each department, it stores information about it's name, income, and manager. The program is able to do various things such as search by name, show employees for a certain department, show manager of a department, etc. It also features some utilities like calculating benefit for each department or for the whole business.

Features

- Modular file storage.
- Simple, fast database access.
- Complete set of tests for every class and functionality.

Setup

Binary installation

- There is a fully working, pre-compiled executable called business-manager ready to download and run after following next steps.
- You have to create an empty /database directory (or alternatively download the sample database folder from here)
- You have to download the help directory if you want to access the help module.
- You can download standard configuration (.config file). If you do not do this you have to add a path for "employee_file" and "department_file" configuration parameters.

Compilation from source code

- All code needed for compilation is included in primary directory. There are no libraries involved other than *C++ std*.
- There is only need to compile *.cpp* files in primary directory. Other classes are for testing purposes.

Compilation with test suite

- Tests are compiled independiently from their own directory.
- Every test depends on *./tests/testConstants.h* to work.
- There are symbolic links to program files to make possible the compilation with one simple command from inside the test directory.
- Tests for EmployeeDAO and DepartmentDAO classes are in the same folder but they should not be compiled together.

Usage

The program has a CLI to access and modify the database as well as to do business operations and utilities with the elements. The commands are the following:

- Help

```
help [ -c <command> ] # show help
```

- Database interaction for employees

```
ls -e # list employees
add -n <name> -s <salary> -d <departmentId> # add employee
edit -i <id> -n <name> -s <salary> -d <departmentId> # edit employee
remove -i <id> # remove employee
find -i <id> # find employee by id
search -n <name> # search employee by name
```

- Database interaction for departments

```
ls -d # list departments
dptadd -n <name> -s <sells> -m <managerId> # add department
dptedit -i <id> -n <name> -s <sells> -m <managerId># edit department
dptremove -i <id> # remove department
dptfind -i <id> # find department by id
dptsearch -n <name> # search department by name
```

- Business methods

```
emps -i <departmentId> # show employees of a certain department
dpt -i <employeeId> # show if employee if manager and managed department if it is
manager -i <departmentId> # shows manager for a certain department
benefit [ -i <departmentId> ] # shows benefit for department or for whole business
```

- Exit program

```
exit # exit the program
quit # same as exit
q # same as exit
```

Configuration

- Program reads it's configuration from a configuration text file. The location of this file can be specified through the first command line argument while calling the program.
- Default location for configuration file is "./.config".
- Configuration parameters can be specified by a "config_param=config_value" pair in a new line.
- Default location for database files are /database/employee_file.txt and /database/department_file.txt
- Default location for help files is under /help directory. Main help file is called help.txt, and the names of the command help files are like: command_help.txt

Room for Improvement

- Output parsing responsibility should be moved from Controller class to a new one.
- Common part from Test classes should be abstracted to AbstractTest to avoid code repetition.
- Command and argument interpretation are too strongly tied to CommandReader internal functions, they could be made more modular to avoid work while adding new functions for the program.

Design patterns used

- Singleton pattern used in Controller class to ensure there is only one instance of Controller to avoid bugs.

- DAO pattern used to encapsulate access to database in it's own layer. Data Access Objects are created in the controller to handle database functioning.

Technologies Used

- C++20

Project Status

Project is complete. Any additions, comments and requests are welcomed in the Issues page, which will be reviewed periodically.

Acknowledgements

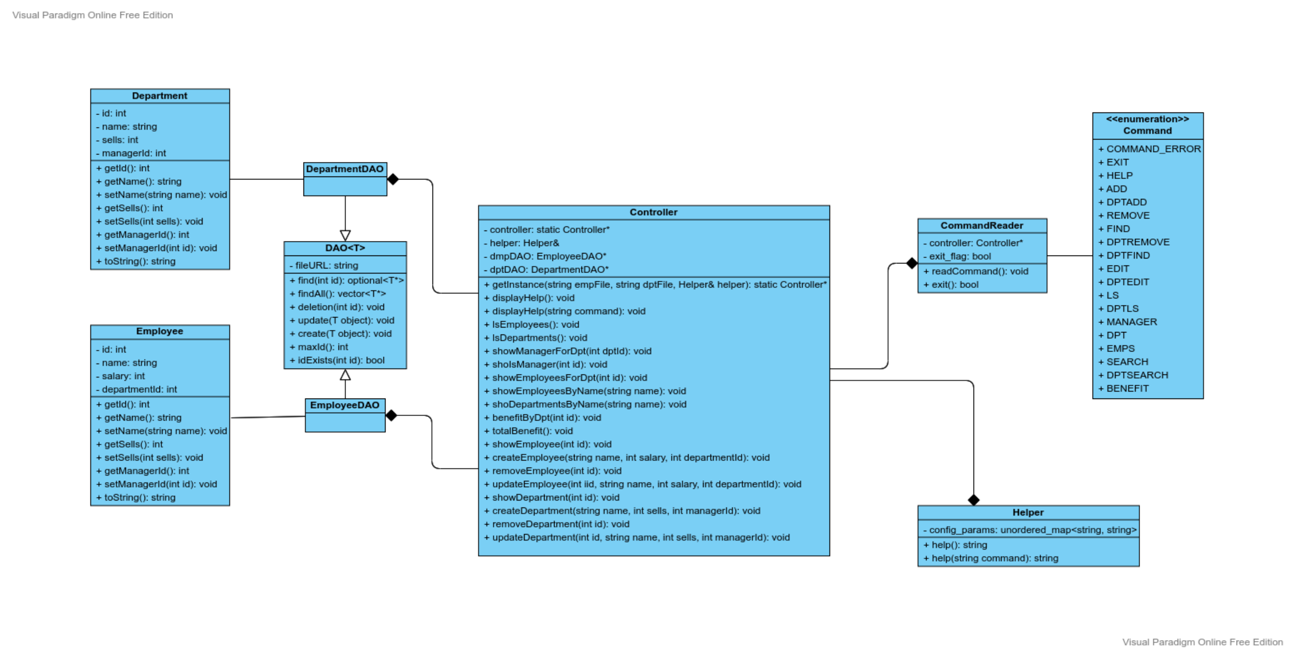
- This program was created as a final project for Object Oriented Programming subject during Erasmus+ program in Politechnika Krakowsa.
- Responsible teacher for subject: Paweł Król

Contact

Created by @adolfo-trocoli LinkedIn profile

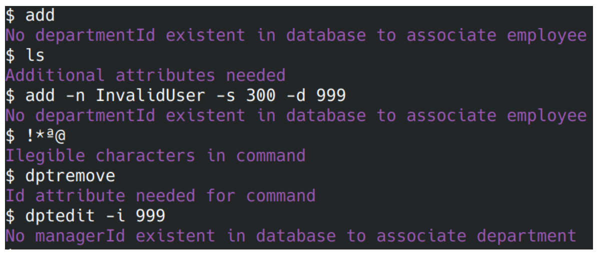
Class diagram

- UML Class diagram with attributes and public methods


Visual Paradigm Online Free Edition
Department
- id: int
- name: string
- sells: int
- managerId: int
+ getId(): int
+ getName(): string
+ setName(string name): void
+ getSells(): int
+ setSells(int sells): void
+ getManagerId(): int
+ setManagerId(int id): void
+ toString(): string
DepartmentDAO
DAO<T>
- fileURL: string
+ find(int id): optional<T*>
+ findAll(): vector<T*>
+ deletion(int id): void
+ update(T object): void
+ create(T object): void
+ maxId(): int
+ idExists(int id): bool
Employee
- id: int
- name: string
- salary: int
- departmentId: int
+ getId(): int
+ getName(): string
+ setName(string name): void
+ getSells(): int
+ setSells(int sells): void
+ getManagerId(): int
+ setManagerId(int id): void
+ toString(): string
EmployeeDAO
Controller
- controller: static Controller*
- helper: Helper&
- dmpDAO: EmployeeDAO*
- dptDAO: DepartmentDAO*
+ getInstance(string empFile, string dptFile, Helper& helper): static Controller*
+ displayHelp(): void
+ displayHelp(string command): void
+ lsEmployees(): void
+ lsDepartments(): void
+ showManagerForDpt(int dptId): void
+ shoIsManager(int id): void
+ showEmployeesForDpt(int id): void
+ showEmployeesByName(string name): void
+ shoDepartmentsByName(string name): void
+ benefitByDpt(int id): void
+ totalBenefit(): void
+ showEmployee(int id): void
+ createEmployee(string name, int salary, int departmentId): void
+ removeEmployee(int id): void
+ updateEmployee(int iid, string name, int salary, int departmentId): void
+ showDepartment(int id): void
+ createDepartment(string name, int sells, int managerId): void
+ removeDepartment(int id): void
+ updateDepartment(int id, string name, int sells, int managerId): void
CommandReader
- controller: Controller*
- exit_flag: bool
+ readCommand(): void
+ exit(): bool
<<enumeration>>
Command
+ COMMAND_ERROR
+ EXIT
+ HELP
+ ADD
+ DPTADD
+ REMOVE
+ FIND
+ DPTREMOVE
+ DPTFIND
+ EDIT
+ DPTEDIT
+ LS
+ DPTLS
+ MANAGER
+ DPT
+ EMPS
+ SEARCH
+ DPTSEARCH
+ BENEFIT
Helper
- config_params: unordered_map<string, string>
+ help(): string
+ help(string command): string
Visual Paradigm Online Free Edition

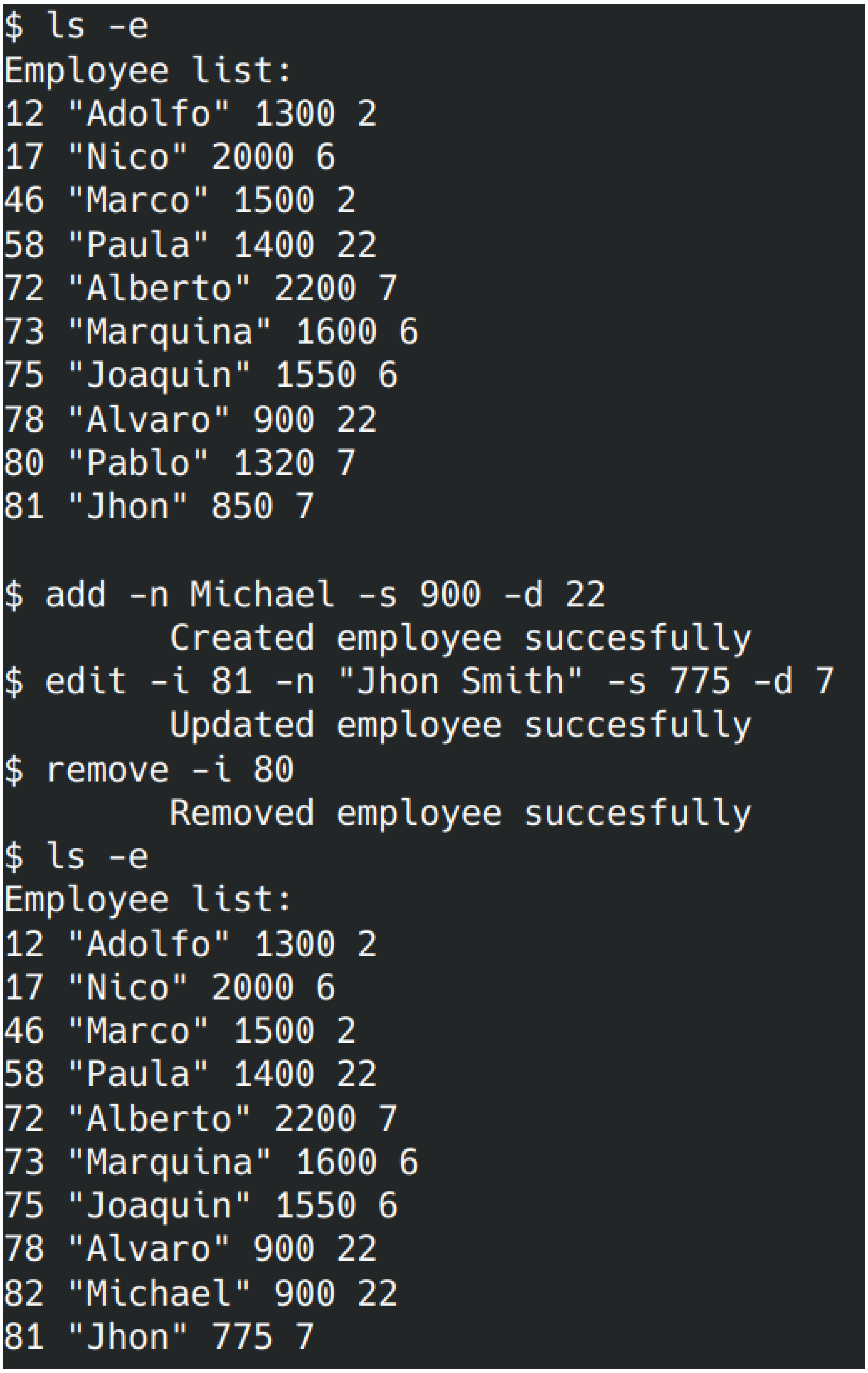
Screenshots

- Some erroneus inputs being handled



```
$ add
No departmentId existent in database to associate employee
$ ls
Additional attributes needed
$ add -n InvalidUser -s 300 -d 999
No departmentId existent in database to associate employee
$ !*ª@
Ilegible characters in command
$ dptremove
Id attribute needed for command
$ dptedit -i 999
No managerId existent in database to associate department
```

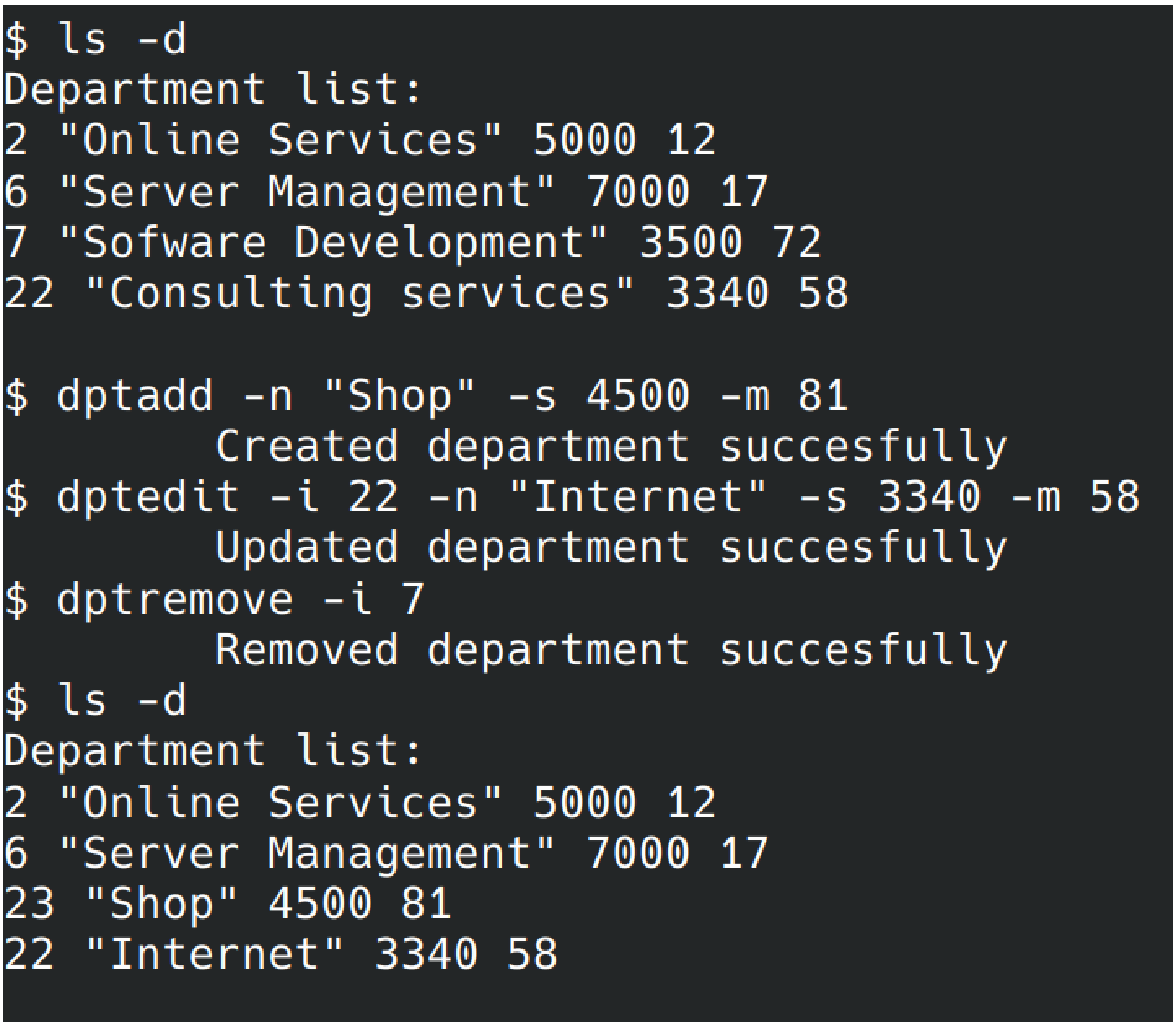
- Example of use of employee management commands



```
$ ls -e
Employee list:
12 "Adolfo" 1300 2
17 "Nico" 2000 6
46 "Marco" 1500 2
58 "Paula" 1400 22
72 "Alberto" 2200 7
73 "Marquina" 1600 6
75 "Joaquin" 1550 6
78 "Alvaro" 900 22
80 "Pablo" 1320 7
81 "Jhon" 850 7

$ add -n Michael -s 900 -d 22
        Created employee succesfully
$ edit -i 81 -n "Jhon Smith" -s 775 -d 7
        Updated employee succesfully
$ remove -i 80
        Removed employee succesfully
$ ls -e
Employee list:
12 "Adolfo" 1300 2
17 "Nico" 2000 6
46 "Marco" 1500 2
58 "Paula" 1400 22
72 "Alberto" 2200 7
73 "Marquina" 1600 6
75 "Joaquin" 1550 6
78 "Alvaro" 900 22
82 "Michael" 900 22
81 "Jhon" 775 7
```

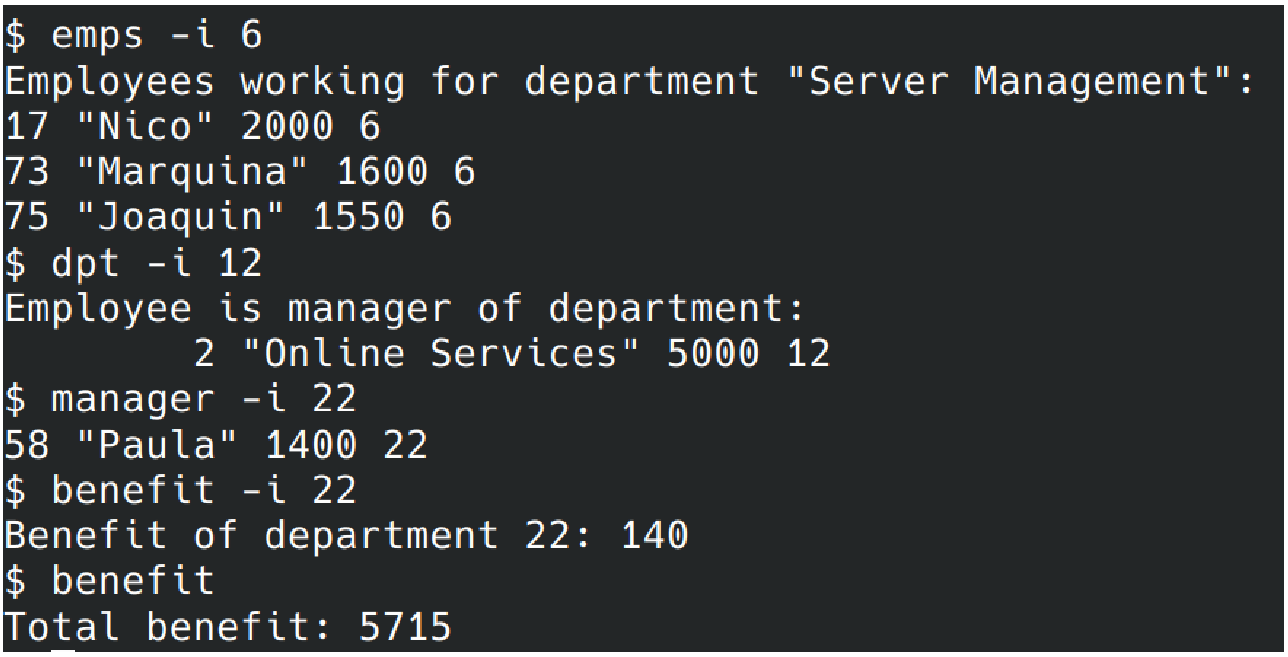
- Example of use of department management commands



```
$ ls -d
Department list:
2 "Online Services" 5000 12
6 "Server Management" 7000 17
7 "Sofware Development" 3500 72
22 "Consulting services" 3340 58

$ dptadd -n "Shop" -s 4500 -m 81
        Created department succesfully
$ dptedit -i 22 -n "Internet" -s 3340 -m 58
        Updated department succesfully
$ dptremove -i 7
        Removed department succesfully
$ ls -d
Department list:
2 "Online Services" 5000 12
6 "Server Management" 7000 17
23 "Shop" 4500 81
22 "Internet" 3340 58
```

- Example of use of business commands



```
$ emps -i 6
Employees working for department "Server Management":
17 "Nico" 2000 6
73 "Marquina" 1600 6
75 "Joaquin" 1550 6
$ dpt -i 12
Employee is manager of department:
        2 "Online Services" 5000 12
$ manager -i 22
58 "Paula" 1400 22
$ benefit -i 22
Benefit of department 22: 140
$ benefit
Total benefit: 5715
```