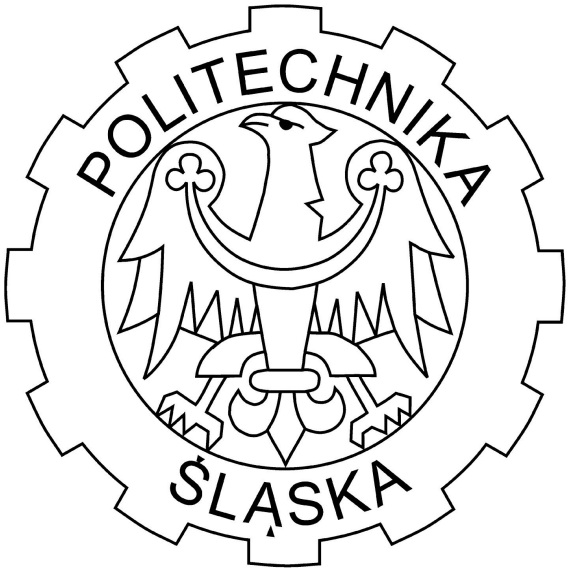
Politechnika Śląska w Gliwicach
Wydział Automatyki, Elektroniki i Informatyki



Programowanie Komputerów semestr IV

Sprawozdanie z projektu

17.04.2020

Autor: Sara Witek
Prowadzący przedmiot: dr inż. Krzysztof Pasterak

Gliwice 2020

Temat projektu:
Baza danych pracowników i firm.

I. Analiza tematu

1. Opis tematu

Zaprojektowany przez mnie projekt to rozszerzona wersja projektu z semestru trzeciego, który umożliwiał obsługę bazy danych pracowników. Projekt opiera się na zasadach działania angielskiej agencji pracy, dlatego wszelkie dane osobowe o pracownikach oraz nazewnictwo są oparte na tym systemie.

Pracowników możemy podzielić na pracowników biurowych oraz pracowników budowlanych. Do klasy pracowników biurowych należy manager, księgowa oraz kierowca, a do klasy pracowników budowlanych kierownik oraz cieśla.

W projekcie możemy rozróżnić podział na firmę wewnętrzną oraz na firmy zewnętrzne. Firma wewnętrzna to firma, na której aktualnie pracujemy. Nasza firma nie posiada własnego miejsca pracy, ale umożliwia zatrudnienie pracowników biurowych. Pracownicy biurowi należą do agencji, więc nie ma możliwości ich wypożyczenia. Zewnętrzne firmy budowlane to firmy, które mogą wypożyczać naszych pracowników budowlanych na swoje miejsca pracy.

W projekt uwzględniono również system płatności pracownikom. Każda firma posiada różne stawki dla każdego rodzaju pracy. Płatności w praktyce są realizowane co tydzień, dlatego kwota do zapłaty zależy od ilości przepracowanych godzin. Pracownicy są traktowani jako poddostawcy usług, więc obowiązek rozliczenia z państwem jest ich obowiązkiem.

2. Wybór klas, bibliotek oraz algorytmów

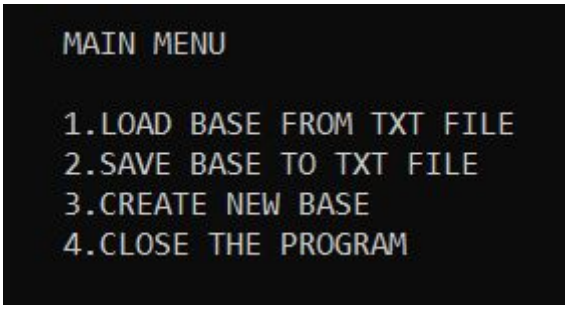
W programie występują cztery główne klasy, które reprezentują odpowiednio pracowników, firmę wewnętrzna, firmy zewnętrzne oraz płatności.
W programie zdecydowałam się na użycie wzorca projektowego typu strategia, ze względu na potrzebę utworzenia oddzielnych interfejsów do obsługi wyżej wymienionych klas głównych.
Do programu zostały dołączone również odpowiednie biblioteki np. regex, typeinfo pozwalające na obsługę wybranych przez mnie technik obiektowych.
Do przechowywania obiektów lub danych o obiektach postanowiłam użyć kontenerów biblioteki STL, ze względu możliwość swobodnego dostępu oraz kontroli nad danymi.

II. Specyfikacja zewnętrzna

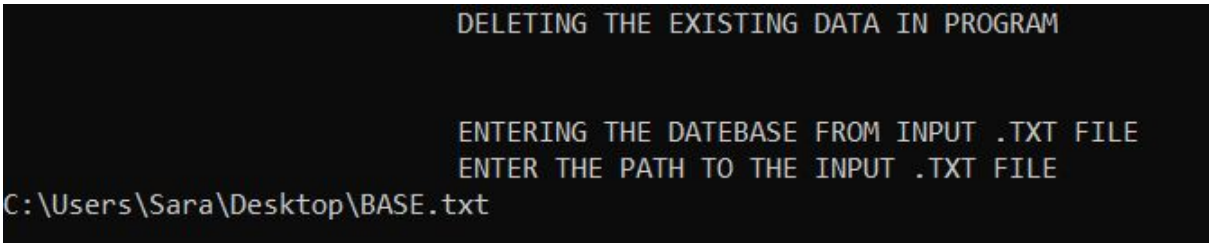
1. Instrukcja działania

Przed rozpoczęciem pracy w programie należy się zapoznać z zasadami panującymi w programie, aby w pełni zrozumieć jego działanie.

Menu główne



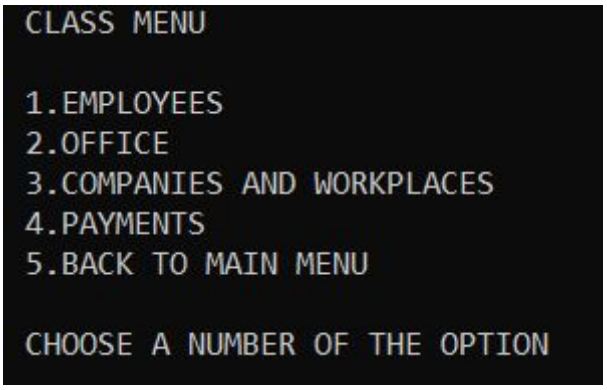
Główne menu programu pozwala na wybranie jednej z opcji rozpoczęcia pracy w naszym programie. Pozwala na wczytanie bazy danych z pliku lub rozpoczęcie nowej bazy danej w zależności od wyboru użytkownika.



W celu wczytania pliku należy podać ścieżkę do pliku txt, aby program poprawnie zadziałał należy używać plików wygenerowanych jedynie przez program.

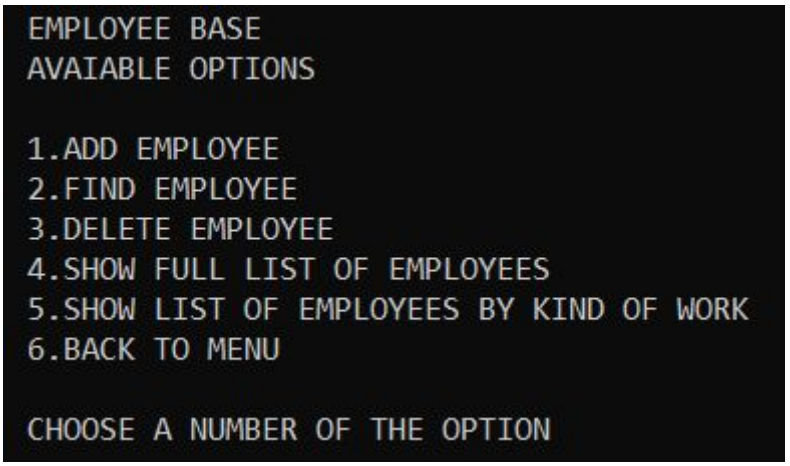
Należy pamiętać, że powrót do menu głównego oznacza zakończenie pracy w naszej bazie danych, dlatego jeżeli chcemy ponownie mieć możliwość zmiany danych należy najpierw zapisać plik i ponownie go załadować.

Menu obsługi



Menu obsługi pozwalające na swobodne przemieszczanie się pomiędzy klasowymi interfejsami.

Menu bazy danych pracowników

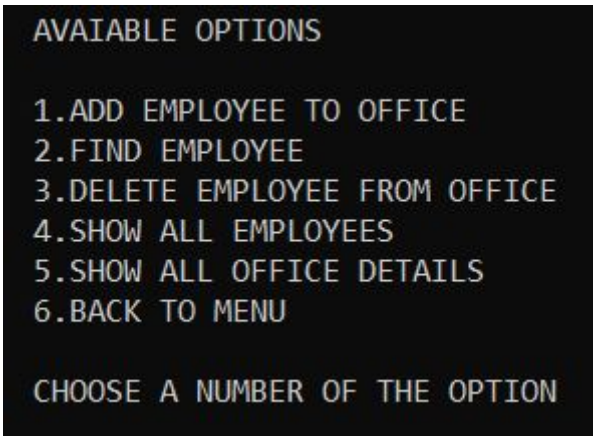


Menu pozwala na dodanie, znalezie lub usunięcie pracownika z bazy danych. Pozwala również na wyświetlanie wszystkich pracowników w bazie danych oraz wyświetlenie po rodzaju pracy.

Należy pamiętać że dodanie pracownika do bazy danych pracowników, dodaje go jedynie do bazy ogólnej. Pracownicy są samozatrudnieni, co oznacza, że nie mają stałej ilości dni i godzin pracy, dlatego mogą być w bazie danych pracowników, ale w danym tygodniu nie pracować.

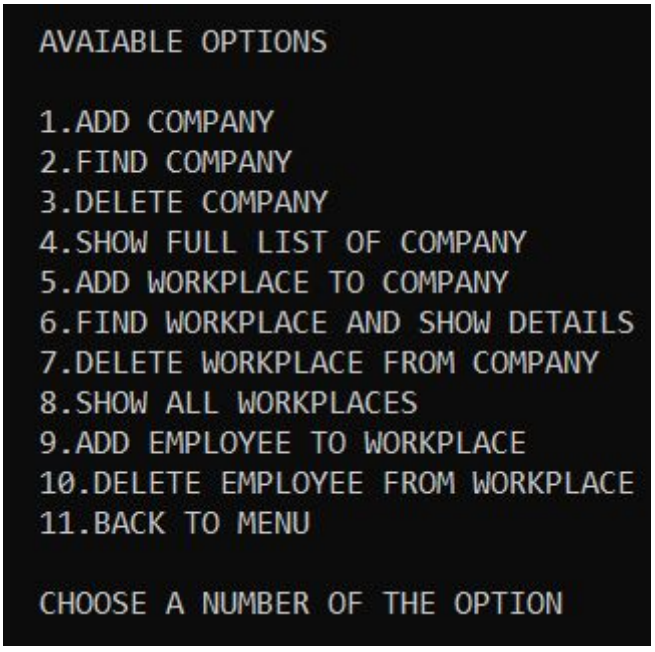
Przypisanie pracownika do biura lub miejsca pracy należy wykonać za pomocą odpowiedniego interfejsu. W programie występuje podział pracowników na biurowych i budowlanych, więc nie jest możliwe dodanie pracownika biurowego do firmy budowlanej.

Menu bazy danych biura



W programie nie ma możliwości dodania kolejnego biura, program umożliwia jedynie edycję bazy danych pracowników biurowych. Pozwala na dodawanie, znajdowanie i usuwanie pracowników biurowych.

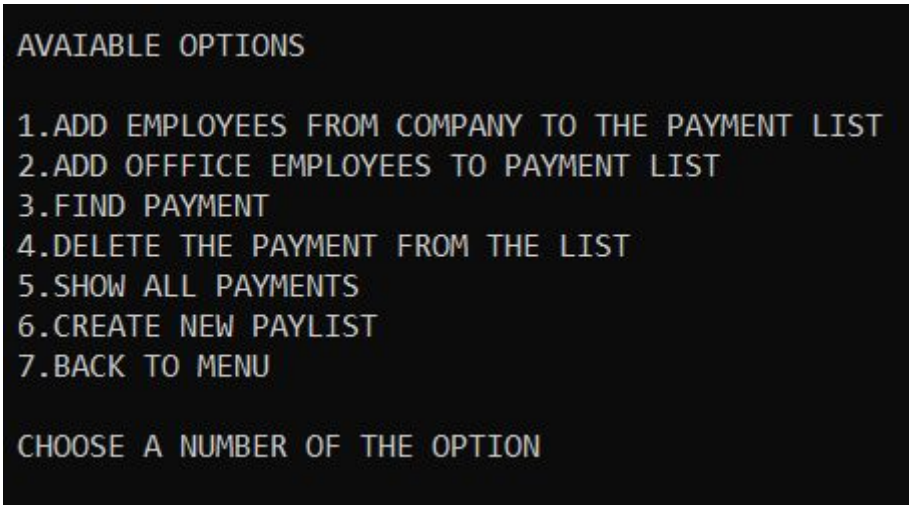
Menu bazy danych firm



.

Menu bazy danych firm pozwalające na obsługę miejsc pracy oraz pracowników budowlanych. Nie ma ograniczeń co do ilości miejsc pracy i firmie. W programie pracownik może być przypisany jedynie do jednego miejsca pracy.

Menu płatności



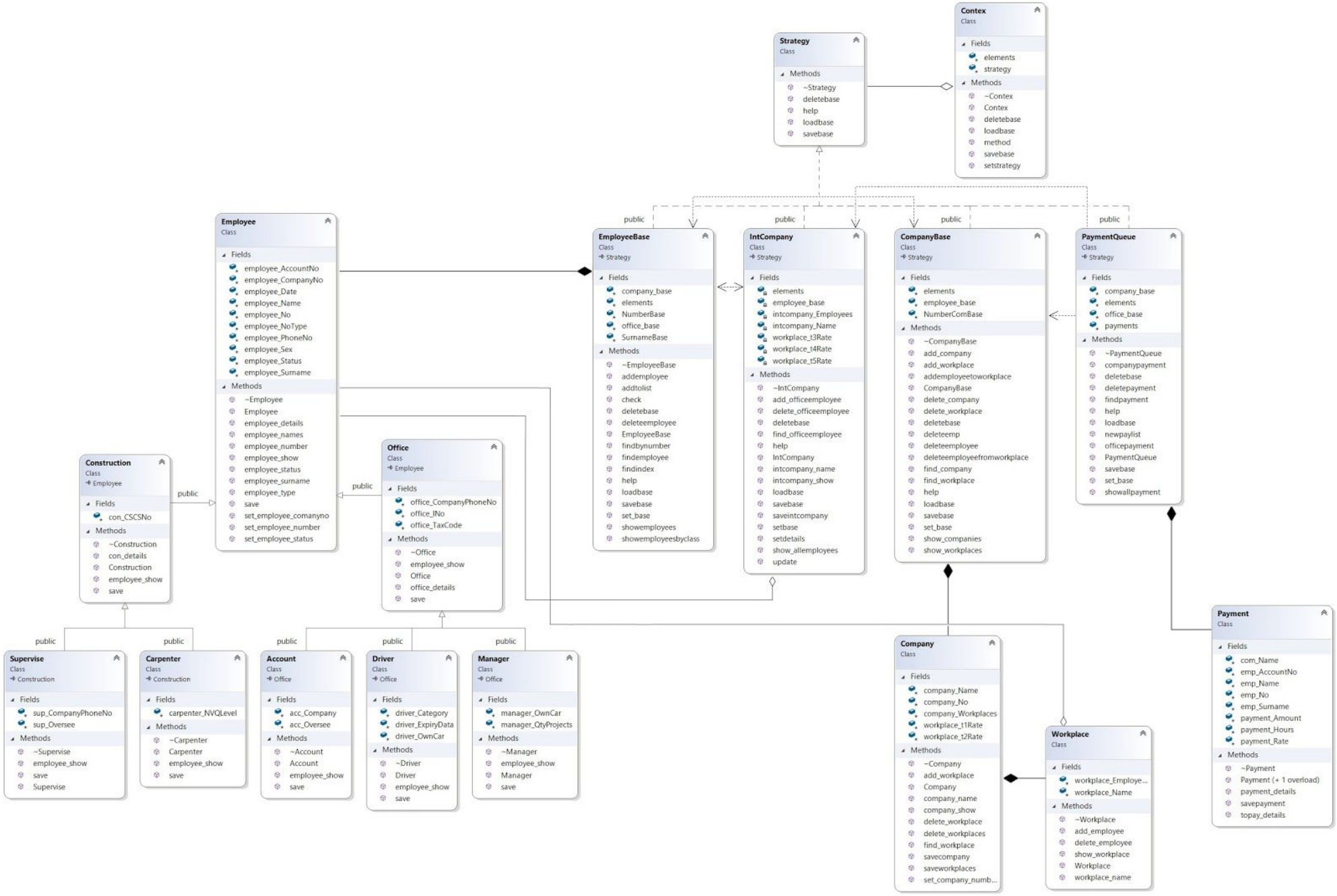
Menu obsługi płatności pozwalające na dodanie, znalezienie lub usunięcie płatności. Płatności funkcjonują jak płatności tygodniowe, dlatego uzupełniamy ilość przepracowanych godzin. Płatności wykonuje się grupowo, czyli jest tylko możliwość dodania do kolejki zapłaty całej firmy. Należy pamiętać że pracownik musi być przypisany do jakiejś firmy bądź biura, aby można było mu zapłacić.

III. Specyfikacja wewnętrzna

1. Diagram klas

2. Powiązania pomiędzy klasami

W programie występują istotne powiązania, które mają wpływ na cykl życia innych klas, bądź są jedynie powiązane relacją zależności. W programie możemy zauważyć implementację wzorca projektowego strategia, w którym poszczególne interfejsy odpowiadają za obsługę danych klas głównych.


Contex
Class
Fields
elements
strategy
Methods
~Contex
Contex
deletebase
loadbase
method
savebase
setstrategy
Strategy
Class
Methods
~Strategy
deletebase
help
loadbase
savebase
public
Employee
Class
Fields
employee_AccountNo
employee_CompanyNo
employee_Date
employee_Name
employee_No
employee_NoType
employee_PhoneNo
employee_Sex
employee_Status
employee_Surname
Methods
~Employee
Employee
employee_details
employee_names
employee_number
employee_show
employee_status
employee_surname
employee_type
save
set_employee_comanyno
set_employee_number
set_employee_status
EmployeeBase
Class
Strategy
Fields
company_base
elements
NumberBase
office_base
SurnameBase
Methods
~EmployeeBase
addemployee
addtolist
check
deletebase
deleteemployee
EmployeeBase
findbynumber
findemployee
findindex
help
loadbase
savebase
set_base
showemployees
showemployeesbyclass
IntCompany
Class
Strategy
Fields
elements
employee_base
intcompany_Employees
intcompany_Name
workplace_t3Rate
workplace_t4Rate
workplace_t5Rate
Methods
~IntCompany
add_officeemployee
delete_officeemployee
deletebase
find_officeemployee
help
IntCompany
intcompany_name
intcompany_show
loadbase
savebase
saveintcompany
setbase
setdetails
show_allemployees
update
CompanyBase
Class
Strategy
Fields
elements
employee_base
NumberComBase
Methods
~CompanyBase
add_company
add_workplace
addemployeetoworkplace
CompanyBase
delete_company
delete_workplace
deletebase
deleteemp
deleteemployee
deleteemployeefromworkplace
find_company
find_workplace
help
loadbase
savebase
set_base
show_companies
show_workplaces
PaymentQueue
Class
Strategy
Fields
company_base
elements
office_base
payments
Methods
~PaymentQueue
companypayment
deletebase
deletepayment
findpayment
help
loadbase
newpaylist
officepayment
PaymentQueue
savebase
set_base
showallpayment
Construction
Class
Employee
Fields
con_CSCSNo
Methods
~Construction
con_details
Construction
employee_show
save
Office
Class
Employee
Fields
office_CompanyPhoneNo
office_INo
office_TaxCode
Methods
~Office
employee_show
Office
office_details
save
Supervise
Class
Construction
Fields
sup_CompanyPhoneNo
sup_Oversee
Methods
~Supervise
employee_show
save
Supervise
Carpenter
Class
Construction
Fields
carpenter_NVQLevel
Methods
~Carpenter
Carpenter
employee_show
save
Account
Class
Office
Fields
acc_Company
acc_Oversee
Methods
~Account
Account
employee_show
save
Driver
Class
Office
Fields
driver_Category
driver_ExpiryData
driver_OwnCar
Methods
~Driver
Driver
employee_show
save
Manager
Class
Office
Fields
manager_OwnCar
manager_QtyProjects
Methods
~Manager
employee_show
Manager
save
Company
Class
Fields
company_Name
company_No
company_Workplaces
workplace_t1Rate
workplace_t2Rate
Methods
~Company
add_workplace
Company
company_name
company_show
delete_workplace
delete_workplaces
find_workplace
savecompany
saveworkplaces
set_company_numb...
Workplace
Class
Fields
workplace_Employe...
workplace_Name
Methods
~Workplace
add_employee
delete_employee
show_workplace
Workplace
workplace_name
Payment
Class
Fields
com_Name
emp_AccountNo
emp_Name
emp_No
emp_Surname
payment_Amount
payment_Hours
payment_Rate
Methods
~Payment
Payment (+ 1 overload)
payment_details
savepayment
topay_details

Class Contex

Klasa pozwalająca na obsługę programu, za pomocą odpowiednio metody *setstrategy()* czyli ustaw odpowiedni interfejs oraz metody *method(),* która wyświetla odpowiednie menu. Użytkownik w zależności od własnych preferencji porusza się pomiędzy głównymi interfejsami.

```
class Contex {
protected:
    Strategy * strategy; // Strategy
    vector <string> elements;
public:
    // Constructor
    Contex() :strategy(nullptr) {};
    // Deconstructor
    ~Contex() { strategy = nullptr; };

    void method(); // Function, which turns a help interface
    void setstrategy(Strategy* pstrategy); // Set a strategy
    void loadbase(vector <string> elements); // Load a base to txt file
    void savebase(std::ostream& outfile); // Save a base from txt file
    void deletebase(); // Deleting the existing base data

    // Friend funtions
    // Load a base to txt file
    friend bool loaddatabase(vector <Strategy*> vec, Contex * contex);
    // Save a base from txt file
    friend bool savedatabase(vector <Strategy*> vec, Contex* contex);
};
```

Class Strategy

Abstrakcyjna klasa, której wszelkie metody są nadpisywane przez elementy dziedziczone. Posiada cztery wirtualne metody pozwalające na odczyt i zapis do pliku, usunięcie bazy danych oraz wyświetlenie menu.

```
// Abstract function for changing the interface
class Strategy {
public:
    bool virtual loadbase (vector <string> elements) = 0; // Load a base from txt file
    bool virtual savebase(std::ostream& outfile) = 0; // Save a base to txt file
    void virtual deletebase() = 0; // Load a base to txt file
    bool virtual help() = 0; // Show help information

    // Deconstructor
    virtual ~Strategy() {};
};
```

Class EmployeeBase

Klasa dziedzicząca po klasie Strategy odpowiadająca za interfejs do obsługi bazy danych pracowników. Metody odziedziczone po klasie bazowej Strategy są odpowiednio nadpisane, co pozwala na korzystanie z nich polimorficznie. Klasa zawiera listę wskaźników na elementy typu Employee w kolejności alfabetycznej oraz w kolejności rosnących numerów identyfikacyjnych. W klasie zdefiniowana jest zależność z klasami IntCompany oraz CompanyBase, ze względu na fakt iż usunięcie pracownika z bazy danych pracowników, automatycznie usuwa tą osobę z bazy danych danej firmy.

```
////////////////////////////////////////////////////////////////////////////////////////////////////
// Class responsible for task management for employee base
class EmployeeBase : public Strategy {

protected:
    std::list <Employee*> SurnameBase; // List in alfabetical order
    std::list <Employee*> NumberBase; // List in ascending order , increasing numbers
    vector <string> elements; // Vector, used during iniciacion of the employee
    IntCompany* office_base;
    CompanyBase* company_base;

public:
    // Constructor
    EmployeeBase();
    // Deconstructor
    ~EmployeeBase();

    // Main functions
    void set_base(IntCompany* poffice ,CompanyBase* pbase); // Set a base
    bool loadbase(vector <string> elements) override; // Load a employee base from a .txt file
    bool savebase(std::ostream& outfile) ; // Save a employee base to a .txt file
    void deletebase(); // Delete a existing data base

    // Funcions available for user
    void addemployee(); // Add a employee to data base
    Employee* findemployee(int number, int& index); // Return employee in a list
    bool deleteemployee(); // Delete a employee from the data base
    bool showemployees(); // Show full list of the employees
    bool showemployeesbyclass(); // Show list of the employees by kind of work

    // Managment function
    bool help() override; // Show help information

    /*****************************************************************************************
    // Additional functions
    void addtolist(); // Function, which add employee do the list
    void check(); // Function which check the all details
    void findindex ( Employee* pptr); // Function, which find a index for new employee
    Employee* findbynumber(int number); // Find employee by his number
    friend class Contex;

};
```

Class Employee

Klasa Employee to klasa abstrakcyjna, która zawiera podstawowe dane osobowe na temat pracownika takie jak imię, nazwisko lub numer konta bankowego. Hierarchia dziedziczenia jest trzypoziomowa, ze względu na podział na pracowników budowlanych (klasa Construction) oraz biurowych (klasa Office). Obie te klasy są również abstrakcyjne i posiadają własne indywidualne zmienne. W ostatniej linii dziedziczenie znajduje się ostateczny podział na klasy wykonywanych zawodów. Po klasie Office dziedziczą klasy Manager, Account oraz Driver, a po klasie Construction klasa Supervise oraz Carpenter.

```cpp
// Abstract class
class Employee
{
protected:
    int employee_No; // Individual employee number
    int employee_NoType; // Number of the type of work
    string employee_Name; // Name of the employee
    string employee_Surname; // Surname of the employee
    string employee_Date; // Date of birth in format DD/MM/YYYY
    bool employee_Sex; // Sex of the employee true for Female, false for Man
    string employee_PhoneNo; // Phone number
    string employee_AccountNo; // Account number
    bool employee_Status; // Status of employee - working somewhere or no
    int employee_CompanyNo; // Number of the company

public:

    // Friendship
    friend class EmployeeBase; // Declaration of friendship with a class EmployeeBase

    friend std::ostream& operator<<(std::ostream& outfile, const Employee& pemployee); // Declaration of friendship with operator << which is overloaded

    // Constructor
    Employee(int pNoType, int pNo, string pName, string pSurname, string pDate, char pSex, string pPhone, string pAccount);

    // Deconstructor
    virtual ~Employee() {};

    // Abstract functions
    void virtual employee_show() = 0; // Show all details

    virtual std::ostream& save(std::ostream& outfile) const = 0; // Save to the file all details

    // Functions

    void employee_details(); // Show all employee details
    void employee_names(); // Show surname and name of the employee
    string employee_surname(); // Return a surname of the employee
    void set_employee_number( int pNumber); // Set a employee number
    void set_employee_status(bool status); // Set a employee status
    void set_employee_comanyno(int nnumber); // Set a company number
    int employee_number(); // Return a employee number
    int employee_type (); // Return a type of employee
    int employee_status(); // Return a status of employee

};
```

Class IntCompany

Klasa IntCompany służy do obsługi firmy wewnętrznej czyli naszej agencji pracy. Obiekt tej klasy tworzy się zaraz po uruchomieniu programu, gdyż w programie jednorazowo inicjalizujemy dane na temat naszej firmy wewnętrznej. Zatrudnienie pracownika w biurze jest równoznaczne z przechowaniem wskaźnika na tego pracownika w wektorze. Klasa ta korzysta z metod dostarczonych przez klasę EmployeeBase, dlatego jest między nimi istotna zależność.

```
class EmployeeBase;
class IntCompany : public Strategy {
private:
    string intcompany_Name;  // Name of the company
    double workplace_t3Rate; // Rate per h for Manager
    double workplace_t4Rate; // Rate per h for Account
    double workplace_t5Rate; // Rate per h for Driver
    vector <Employee*> intcompany_Employees; // Base of employees working on internal company
    vector <string> elements; // Vector, used during iniciacion of the employee
    EmployeeBase* employee_base;

public:
    // Constructor
    IntCompany();
    // Deconstructor
    ~IntCompany();

    // Main functions
    bool loadbase(vector <string> elements) override; // Load a company base from a .txt file
    bool savebase(std::ostream& outfile) override; // Save a company base to a .txt file
    void deletebase() override; // Delete a existing data base

    // Functions avaiable for user
    void setdetails(string pName, double pt3Rate, double pt4Rate, double pt5Rate);
    void setbase(EmployeeBase* pemployee_base); // Set a base
    string intcompany_name(); // Return a internal company name
    void add_officeemployee(); // Add a employee to office base
    Employee* find_officeemployee(int number); // Find a employee
    void show_allemployees(); // Show all office employees
    void delete_officeemployee(); // Delete a employee from office base
    void intcompany_show(); // Show all internal company details

    void update(int pnumber);// Update after deleting a employee
    //Managment function
    bool help() override; // Show help information

    std::ostream& saveintcompany(std::ostream& outfile) const ; // Save to the file all details
    friend std::ostream& operator<<(std::ostream& outfile, const IntCompany& pcompany); // Declaration of friendship with operator << which is overloaded
    friend class PaymentQueue;
};
```

Class Company

Klasa pozwalająca na obsługę firm zewnętrznych, czyli firm budowlanych oraz ich miejsc pracy. Każdy z obiektów klasy Company posiada wektor wskaźników na klasę Workplace, czyli na miejsce pracy. Miejsca pracy posiada również wektor wskaźników na klasę Employee tak jak to było w przypadku klasy IntCompany. Warto wspomnieć, że istnienie klasy Workplace jest zależne od istnienia klasy Company. Usunięcie miejsca pracy, bądź firmy nie ma wpływu na bazę danych pracowników, jedynie zmienia się stan pracy pracownika na “niepracujący”.

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// ExtCompany class
class Company {

protected:
    int company_No;
    // Number of the company. Different numbers for external companies - construction employees.
    string company_Name;  // Name of the company
    double workplace_t1Rate; // Rate per h for Supervise
    double workplace_t2Rate; // Rate per h for Carpenter
    vector <Workplace*> company_Workplaces; // Base of workplaces

public:

    // Constructor
    Company(string pName, double pt1Rate, double pt2Rate);
    // Deconstuctor
    ~Company();

    std::ostream& savecompany(std::ostream& outfile) ; // Save to the file all company details
    std::ostream& saveworkplaces(std::ostream& outfile);

    friend std::ostream& operator<<(std::ostream& outfile, Company& pcompany); // Declaration of friendship with operator << which is overloaded

    // Main functions
    string company_name(); // Return a company name
    void set_company_number(int pnumber); // Set a company number
    void company_show(); // Show all company details


    // Workplace functions
    void add_workplace(Workplace* pworkplace); // Function which add a workplace
    Workplace* find_workplace(int number); // Function which finding a workplace
    void delete_workplace(vector <Workplace*>::iterator pitr, int pindex);
    void delete_workplaces(); // Function which delete a workplace

    // Friend classes
    friend class Workplace;
    friend class CompanyBase;
    friend class PaymentQueue;
};
```

Class PaymentQueue

Klasa pozwalająca na obsługę płatności pracownikom za przepracowany czas pracy. Klasa ta umożliwia płatność jednocześnie wszystkim pracownikom zarejestrowanym w danej firmie poprzez kolejno dodanie wskaźnika typu Payment do listy płatności. Obiekty typu Payment zawierają zmienne inicjalizowane danymi uzyskanymi z klas Company oraz Employee. Klasa ta jest niezależna od klasy Employee, więc usunięcie pracownika z bazy pracowników nie powoduje usunięcia jego płatności. Klasa ta korzysta z zależności z klasami CompanyBase i IntCompany w celu dodania płatności do listy.

```
// Queue of payments class
class PaymentQueue : public Strategy
{
protected:
    std::list <Payment*> payments; // Queue of payments to do
    vector <string> elements; // Vector, used during iniciacion of the payments
    CompanyBase* company_base;
    IntCompany* office_base;
public:
    // Constructor
    PaymentQueue();
    // Deconstuctor
    ~PaymentQueue();

    // Main functions
    bool loadbase(vector <string> elements) override; // Load a payment base from a .txt file
    bool savebase(std::ostream& outfile) override; // Save a payment base to a .txt file
    void deletebase() override; // Delete the existing payment base

    // Functions avaiable for user
    void set_base(CompanyBase* pbase, IntCompany* obase); // Set a base
    void companypayment(); // Adding employees from company to the payment list
    void officepayment(); // Adding employee from office to the payment list
    Payment* findpayment(int number, int& index); // Find a payment in list
    void deletepayment(); // Delete a payment from the list
    void showallpayment(); // Show full list of payment
    void newpaylist(); // Create new paylist

    //Managment function
    bool help() override;  // Show help information
};
```

3. Techniki obiektowe

W stosunku do projektu z trzeciego semestru została wprowadzona znaczna modyfikacja, gdyż wszelkie struktury danych zostały zamienione na ich odpowiedniki w kontenerach STL. W programie zastosowano na szeroką skalę kontenery list oraz vector, a w celu ułatwienia ich obsługi zastosowane zostały również iteratory STL.
W programie pełni również istotna rolę mechanizm RTTI, ze względu na fakt posiadania klas abstrakcyjnych i dziedziczenia. Korzystanie z narzędzia dynamic_cast pozwoliło na rozróżnienie danych klas pracowników, poprzez rzutowanie wskaźnika na klasę bazową na wskaźnik do obiektu klasy pochodnej.
Ostatnią zastosowaną techniką jest wyrażenie regularne regex, które pozwoliło mi na rozwiązanie problemu odczytu danych w pliku. W programie występuje istotna zależność pomiędzy klasą Company, a klasa Workplace. W momencie odczytu należy utworzyć obiekt Company, a następnie przypisać wszelkie miejsca pracy czyli obiekty Workplace do obiektu Company. Wyrażenie te pozwoliło mi na jednoczesny odczyt wszystkich danych o firmie, a następnie na odpowiednie rozróżnienie i podział w celu poprawnej implementacji.

IV. Testowanie i uruchamianie

W trakcie tworzenia programu natrafiłam na komplikacje z powodu dużej ilości zależności jakie występują pomiędzy klasami. Aby uniknąć nieporozumień wprowadzone zostały bardziej restrykcyjne warunki np: możliwość pracy tylko w jednym miejscu.
W programie, ze względu na fakt, że to użytkownik wprowadza dane nie ma możliwości sprawdzenia ich całkowitej poprawności. W programie stworzyłam dodatkowe funkcje, które uniemożliwiają wpisywania innych znaków niż te oczekiwane. Warto dodać, że pojawiły się również drobne błędy logiczne z mojej strony takie jak przeoczenie możliwości wpisywania daty późniejszej niż rok 2019.