MVM

Házi Feladat

Specifikáció

Jajecnik Marcell- DMB3AD

2023. 04. 24.

Feladat leírás

Tervezze meg a Meseországi Villamos Művek (MVM) nyilvántartási rendszerének egyszerűsített objektummodelljét, majd valósítsa azt meg! A rendszerrel minimum a következő műveleteket kívánjuk elvégezni:

- ügyfél adatinak felvétele
- szolgáltatási szerződés kötése
- szolgáltatási díj előírása (számlázás)
- szolgáltatási díj befizetése
- egyenleg lekérdezése
- fogyasztás bejelentése

A rendszer lehet bővebb funkcionalitású, ezért nagyon fontos, hogy jól határozza meg az objektumokat és azok felelősségét. Demonstrálja a működést külön modulként fordított tesztprogrammal! A megoldáshoz ne használjon STL tárolót!

Bemenetek

Forras.txt fájl ami tartalmazza az ügyfél adatait a következő sorrendben

- ügyfél neve
- ügyfél típus
- szerződés azonosító
- szerződéstípus (enum)
- szerződött időszak
- tarifa
- egyenleg
- fogyasztás

Adatfelvitel parancssorból, egy információ, 1 sor

- ügyfél neve
- ügyfél típus
- szerződéstípus (enum)
- szerződéshossz

- tarifa
- egyenleg
- fogyasztás

Tesztesetek futtatása fájlból

- egyenleglekérdezés
- számlakészítés
- befizetés
- fogyasztás bejelentése

Kimenetek

Számlakészítés fájlba

- Kinek a számlája
- Milyen időszakról
- Mekkora összeg

Számlakészítés parancssorba

- Kinek a számlája
- Milyen időszakról
- Mekkora összeg

Egyenleglekérdezés

- Függvényvisszaadja a jelenlegi egyenleget

Teszt tesztesetekkel tesztfájlal

MVM

Házi Feladat

Terv

Jajecnik Marcell- DMB3AD

2023. 05. 03.

A program struktúrája

A program osztálydiagramja, a fontosabb adattagokkal és függvényekkel

- A person és a company osztályok publikus módon öröklődnek a client-ből
- A myvektor generikus osztály az indexelhetőséghez
- A myvektor amennyiben betelik az eddigi méretének dupláját foglalja
- Egy emberhez több szerződés is kapcsolódhat

MVM_vegso

Generated by Doxygen 1.9.6

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

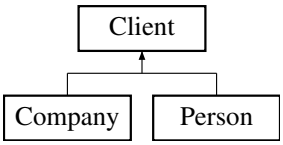
Class Documentation

4.1 Client Class Reference

Ügyfél adatait tárolja.

`#include <clients.h>`

Inheritance diagram for Client:



Public Member Functions

- Client (const std::string name_in)
  Alapértékes kontruktor.
- size_t getSumOfAll_Contract ()
  Visszaadja a kliens által eddig kötött szerződések számát.
- std::string getName () const
  Visszadja a kliens nevét.
- void addContract (Contract &contract_in)
  Szerződés hozzáadása a mycontracts hez.
- virtual std::string getsub_Client_info ()
  Visszaadja a gyermek osztály személyes adatait kiíráshoz egy string-ben.
- void save_invoice (size_t contract_index, const Date &date)
  Számlát állít ki fájlba.
- Contract & getContract (size_t contract_ID)
  Visszaadja azt a szerződést aminek az azonosítóját megadtuk.
- virtual ~Client ()
  Virtuális destruktor.

4.1.1 Detailed Description

Ügyfél adatait tárolja.

Parameters

name	Az ügyfél neve
mycontracts	Az ügyfél szerződéseit tároló tömb
SumOfAll_contracts	Azt tárolja hogy a kliens megléte alatt összesen hány szerződést kötött (A szerződés azonosítására használ a felhasználó számára)

Definition at line 21 of file clients.h.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 Client()

```
Client::Client (
            const std::string name_in )  [inline]
```

Alapértékes kontruktor.

Parameters

name_in	Ügyfél neve
sumofContracts	A kliens által eddig kötött szerződések száma (nem jelenlegi)

Definition at line 33 of file clients.h.

4.1.2.2 ~Client()

```
virtual Client::~Client ( )  [inline], [virtual]
```

Virtuális destruktor.

Definition at line 77 of file clients.h.

4.1.3 Member Function Documentation

4.1.3.1 addContract()

```
void Client::addContract (
            Contract & contract_in )
```

Szerződés hozzáadása a mycontracts hez.

Parameters

contract↩ _in	a hozzáadandó szerződésre mutató referencia.

Definition at line 11 of file clients.cpp.

4.1.3.2 getContract()

```
Contract & Client::getContract (
            size_t contract_ID )
```

Visszaadja azt a szerződést aminek az azonosítóját megadtuk.

Parameters

contract_ID	A szerződés azonosítója

Returns

Definition at line 28 of file clients.cpp.

4.1.3.3 getName()

```
std::string Client::getName ( ) const
```

Visszadja a kliens nevét.

Returns

kliens neve

Definition at line 7 of file clients.cpp.

4.1.3.4 getsub_Client_info()

```
std::string Client::getsub_Client_info ( )  [virtual]
```

Visszaadja a gyermek osztály személyes adatait kiíráshoz egy string-ben.

Returns

gyermek osztály személyes adatai kiíráshoz egy string-ben

Reimplemented in Person, and Company.

Definition at line 20 of file clients.cpp.

4.1.3.5 getSumOfAll_Contract()

```
size_t Client::getSumOfAll_Contract ( )
```

Visszaadja a kliens által eddig kötött szerződések számát.

Returns

kliens által eddig kötött szerződések száma

Definition at line 16 of file clients.cpp.

4.1.3.6 save_invoice()

```
void Client::save_invoice (
            size_t contract_index,
            const Date & date )
```

Számlát állít ki fájlba.

Parameters

contract_index	A szerződés sorszáma a tárolt szerződések között

Definition at line 24 of file clients.cpp.

The documentation for this class was generated from the following files:

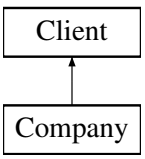
- clients.h
- clients.cpp

4.2 Company Class Reference

Egy cég adatait tárolja.

```
#include <clients.h>
```

Inheritance diagram for Company:



Public Member Functions

- Company (std::string name_in="", std::string ID_in="")

  Alapértékes kontruktor.
- void setID (std::string ID_in)

  Cég adószámának állítása.
- std::string getID () const

  Cég adószámát adja vissza.
- std::string getsub_Client_info ()

  Visszaadja a gyermek osztály személyes adatait kiíráshoz egy string-ben.
- ~Company ()

  Destruktor.

Public Member Functions inherited from Client

- Client (const std::string name_in)

  Alapértékes kontruktor.
- size_t getSumOfAll_Contract ()

  Visszaadja a kliens által eddig kötött szerződések számát.
- std::string getName () const

  Visszadja a kliens nevét.
- void addContract (Contract &contract_in)

  Szerződés hozzáadása a mycontracts hez.
- virtual std::string getsub_Client_info ()

  Visszaadja a gyermek osztály személyes adatait kiíráshoz egy string-ben.
- void save_invoice (size_t contract_index, const Date &date)

  Számlát állít ki fájlba.
- Contract & getContract (size_t contract_ID)

  Visszaadja azt a szerződést aminek az azonosítóját megadtuk.
- virtual ~Client ()

  Virtuális destruktor.

4.2.1 Detailed Description

Egy cég adatait tárolja.

Parameters

Tax_ID	A cég adóazonosító száma 11 jegyű

Definition at line 124 of file clients.h.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 Company()

```
Company::Company (
            std::string name_in = "",
            std::string ID_in = "" )  [inline]
```

Alapértékes kontruktor.

Parameters

name_in	Bejövő Cégnév
ID_in	Bejövő Cég adószám
SumOfAllContract	A kliens által eddig kötött szerződések száma (nem jelenlegi)

Definition at line 135 of file clients.h.

4.2.2.2 ~Company()

```
Company::~Company ( )
```

Destruktor.

Definition at line 64 of file clients.cpp.

4.2.3 Member Function Documentation

4.2.3.1 getID()

```
std::string Company::getID ( ) const
```

Cég adószámát adja vissza.

Returns

Cég adószáma

Definition at line 52 of file clients.cpp.

4.2.3.2 getsub_Client_info()

```
std::string Company::getsub_Client_info ( )  [virtual]
```

Visszaadja a gyermek osztály személyes adatait kiíráshoz egy string-ben.

Returns

gyermek osztály személyes adatai kiíráshoz egy string-ben

Reimplemented from Client.

Definition at line 60 of file clients.cpp.

4.2.3.3 setID()

```
void Company::setID (
            std::string ID_in )
```

Cég adószámának állítása.

Parameters

ID_in	Bejövő Cég adószám

Definition at line 56 of file clients.cpp.

The documentation for this class was generated from the following files:

- clients.h
- clients.cpp

4.3 Contract Class Reference

Egy szerződést tárol.

```
#include <contracts.h>
```

Public Member Functions

- Contract ()

  Paraméter nélküli konstruktor.
- Contract (Client *client_in, ContractType ctype_in, int year_begin, int month_begin, int day_begin, int year_end, int month_end, int day_end, int invo_in_year, int invo_in_month, int invo_in_day, size_t ContractID_in, double tariff_in=0.0, double balance_in=0.0, double consumption_in=0.0)

Alapértékes konstruktor egy szerződéshez.

- void setCtype (ContractType ctype_in)

  Beállítja a szerződés típusát.
- void setTariff (double tariff_in)

  Beállítja az új tarifát.
- void setBalance (double balance_in)

  Beállítja a jelenlegi egyenleget.
- void addConsumption (double con_in)

  Hozzáad a fogyasztott mennyiséghez.
- void setLast_invoicing (Date last_in)

  Beállítja a legutolsó számlázás idejét.
- void setCtime (Contract_date ctime_in)

  Beállítja a szerződés időtartományát.
- Client ∗ getClient ()

  Visszaadja az ügyfél adatait.
- size_t getContractID () const

  Visszaadja a szerződés azonosítóját.
- ContractType getCtype () const

  Visszaadja a szerződés típusát.
- double getTariff () const

  Visszaadja a tarifát.
- double getBalance () const

  Visszaadja a jelenlegi egyenleget.
- double getConsumption () const

  Visszaadja az eddigi fogyasztást a legutóbbi számla kiállítása óta.
- Contract_date getCtime () const

  Visszaadja a szerződés időtartományát.
- const std::string Ctype_toString () const

  A szerződés típusát adja vissza stringként.
- Date getLast_invoicing () const

  Visszaadja a legutótóbbi számla kiállításának az időpontját.
- void pay (double amount)

  Befizetés egyenlegrendezéshez.
- void invoice (const Date &today)

  Számla kiállítása fájlba.
- ∼Contract ()

  Destruktor.

4.3.1 Detailed Description

Egy szerződést tárol.

Parameters

client	Tárolja a szerződéstkötő adatait
contract_ID	A szerződés azonosítószáma
contract_type	A szerződés típusát tárolja
contract_time	A szerződés időtartamát adja meg egy kezdő és egy végdátummal
last_invoicing	Utolsó számlakibocsájtás ideje
tariff	A KWh-kénti árat adja meg
balance	Tárolja a szerződéshez tartozó számla egyenlegét
consumption	Tárolja a fogyasztást a legutóbbi számlakibocsájtás óta

Definition at line 41 of file contracts.h.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 Contract() [1/2]

```
Contract::Contract ( )  [inline]
```

Paraméter nélküli konstruktor.

Definition at line 48 of file contracts.h.

4.3.2.2 Contract() [2/2]

```
Contract::Contract (
            Client * client_in,
            ContractType ctype_in,
            int year_begin,
            int month_begin,
            int day_begin,
            int year_end,
            int month_end,
            int day_end,
            int invo_in_year,
            int invo_in_month,
            int invo_in_day,
            size_t ContractID_in,
            double tariff_in = 0.0,
            double balance_in = 0.0,
            double consumption_in = 0.0 )  [inline]
```

Alapértékes konstruktor egy szerződéshez.

Parameters

client_in	Szerződést kötő ügyfél (kötelező)
ctype_in	Szerződés típusa
year_begin	Szerződés kezdeti éve
month_begin	Szerződés kezdeti hónapja
day_begin	Szerződés kezdeti napja
year_end	Szerződés végső éve
month_end	Szerződés végső hónapja
day_end	Szerződés végső napja
invo_in_year	Legutóbbi számlázás éve
invo_in_month	Legutóbbi számlázás hónapja
invo_in_day	Legutóbbi számlázás napja
ContractID_in	A szerződés azonosítószáma
tariff_in	KWh-kénti ár
balance_in	Jelenlegi egyenleg
consumption_in	Jelenlegi fogyasztás

Definition at line 68 of file contracts.h.

4.3.2.3 ~Contract()

```
Contract::~Contract ( )
```

Destruktor.

Definition at line 124 of file contracts.cpp.

4.3.3 Member Function Documentation

4.3.3.1 addConsumption()

```
void Contract::addConsumption (
            double con_in )
```

Hozzáad a fogyasztott mennyiséghez.

Parameters

con_in	Bejövő fogyasztás

Definition at line 19 of file contracts.cpp.

4.3.3.2 Ctype_toString()

```
const std::string Contract::Ctype_toString ( ) const
```

A szerződés típusát adja vissza stringként.

Returns

szerződés típusa adja vissza stringként

Definition at line 59 of file contracts.cpp.

4.3.3.3 getBalance()

```
double Contract::getBalance ( ) const
```

Visszaadja a jelenlegi egyenleget.

Returns

Jelenlegi egyenleg

Definition at line 47 of file contracts.cpp.

4.3.3.4 getClient()

```
Client * Contract::getClient ( )
```

Visszaadja az ügyfél adatait.

Returns

Ügyfél adatai

Definition at line 31 of file contracts.cpp.

4.3.3.5 getConsumption()

```
double Contract::getConsumption ( ) const
```

Visszaadja az eddigi fogyasztást a legutóbbi számla kiállítása óta.

Returns

Eddigi fogyasztás

Definition at line 51 of file contracts.cpp.

4.3.3.6 getContractID()

```
size_t Contract::getContractID ( ) const
```

Visszaadja a szerződés azonosítóját.

Returns

szerződés azonosítója

Definition at line 35 of file contracts.cpp.

4.3.3.7 getCtime()

```
Contract_date Contract::getCtime ( ) const
```

Visszaadja a szerződés időtartományát.

Returns

Szerződés időtartománya

Definition at line 55 of file contracts.cpp.

4.3.3.8 getCtype()

```
ContractType Contract::getCtype ( ) const
```

Visszaadja a szerződés típusát.

Returns

Szerződés típusa

Definition at line 39 of file contracts.cpp.

4.3.3.9 getLast_invoicing()

```
Date Contract::getLast_invoicing ( ) const
```

Visszaadja a legutótóbbi számla kiállításának az időpontját.

Returns

Legutótóbbi számla kiállításának az időpontja

Definition at line 73 of file contracts.cpp.

4.3.3.10 getTariff()

```
double Contract::getTariff ( ) const
```

Visszaadja a tarifát.

Returns

Jelenlegi tarifa

Definition at line 43 of file contracts.cpp.

4.3.3.11 invoice()

```
void Contract::invoice (
            const Date & today )
```

Számla kiállítása fájlba.

Parameters

today	számla kiállításának napja (név amiatt mert a számlakiállítás napja általában a jelenlegi nappal egyezik meg)

Definition at line 81 of file contracts.cpp.

4.3.3.12 pay()

```
void Contract::pay (
            double amount )
```

Befizetés egyenlegrendezéshez.

Parameters

amount	Befizetett összeg

Definition at line 77 of file contracts.cpp.

4.3.3.13 setBalance()

```
void Contract::setBalance (
            double balance_in )
```

Beállítja a jelenlegi egyenleget.

Parameters

balance_in	Beállítandó egyenleg

Definition at line 15 of file contracts.cpp.

4.3.3.14 setCtime()

```
void Contract::setCtime (
            Contract_date ctime_in )
```

Beállítja a szerződés időtartományát.

Parameters

ctime↩ _in	Bejövő időtartomány

Definition at line 27 of file contracts.cpp.

4.3.3.15 setCtype()

```
void Contract::setCtype (
            ContractType ctype_in )
```

Beállítja a szerződés típusát.

Parameters

ctype↩ _in	Beállítandó szerződéstípus

Definition at line 7 of file contracts.cpp.

4.3.3.16 setLast_invoicing()

```
void Contract::setLast_invoicing (
            Date last_in )
```

Beállítja a legutolsó számlázás idejét.

Parameters

last↩ _in	Számlázás időpontja

Definition at line 23 of file contracts.cpp.

4.3.3.17 setTariff()

```
void Contract::setTariff (
            double tariff_in )
```

Beállítja az új tarifát.

Parameters

tariff↩ _in	Beállítandó tarifa

Definition at line 11 of file contracts.cpp.

The documentation for this class was generated from the following files:

- contracts.h
- contracts.cpp

4.4 Contract_date Class Reference

Egy időpárt tárol.

`#include <date.h>`

Public Member Functions

- Contract_date ()
  paraméter nélkül hívható konstruktor
- Contract_date (Date begin_date, Date end_date)
  Konstruktor kezdő és végdátum megadásával.
- Contract_date (int year_begin, int month_begin, int day_begin, int year_end, int month_end, int day_end)
  Konstruktor intekkel megadott paraméterekből.
- Date getBegin () const
  Visszaadja a kezdeti időpontot.
- Date getEnd () const
  Visszaadja a végső időpontot.
- void setBegin (Date beg_in)
  Beállítja a kezdeti időpontot Date-ből.
- void setBegin (int year, int month, int day)
  Beállítja a kezdeti időpontot intekből.
- void setEnd (Date end_in)
  Beállítja a végső időpontot Date-ből.
- void setEnd (int year, int month, int day)
  Beállítja a végső időpontot intekből.
- bool contains (const Date &date_in) const
  Ellenőrzi hogy a paraméterként kapott dátum a dátumpár között van-e.
- Contract_date & operator= (const Contract_date &contractDate_in)
  = operátor az = jel helyes működéséhez

4.4.1 Detailed Description

Egy időpárt tárol.

Parameters

beging	Az kezdeti időpont
end	A végső időpont

Definition at line 101 of file date.h.

4.4.2 Constructor & Destructor Documentation

4.4.2.1 Contract_date() [1/3]

```
Contract_date::Contract_date ( )  [inline]
```

paraméter nélkül hívható konstruktor

Definition at line 111 of file date.h.

4.4.2.2 Contract_date() [2/3]

```
Contract_date::Contract_date (
            Date begin_date,
            Date end_date )  [inline]
```

Konstruktor kezdő és végdátum megadásával.

Parameters

begin_date	Kezdődátum
end_date	Végdátum

Definition at line 119 of file date.h.

4.4.2.3 Contract_date() [3/3]

```
Contract_date::Contract_date (
            int year_begin,
            int month_begin,
            int day_begin,
            int year_end,
            int month_end,
            int day_end )  [inline]
```

Konstruktor intekkel megadott paraméterekből.

Parameters

year_begin	Bejövő kezdeti év
month_begin	Bejövő kezdeti hónap
day_begin	Bejövő kezdeti nap
year_end	Bejövő végzeti év
month_end	Bejövő végzeti hónap
day_end	Bejövő végzeti nap

Definition at line 132 of file date.h.

4.4.3 Member Function Documentation

4.4.3.1 contains()

```
bool Contract_date::contains (
            const Date & date_in ) const
```

Ellenőrzi hogy a paraméterként kapott dátum a dátumpár között van-e.

Parameters

date	paraméterként kapott dátum

Returns

A dátumpár között van?

Definition at line 93 of file date.cpp.

4.4.3.2 getBegin()

```
Date Contract_date::getBegin ( ) const
```

Visszaadja a kezdeti időpontot.

Returns

kezdeti időpont

Definition at line 63 of file date.cpp.

4.4.3.3 getEnd()

```
Date Contract_date::getEnd ( ) const
```

Visszaadja a végső időpontot.

Returns

Végső időpont

Definition at line 67 of file date.cpp.

4.4.3.4 operator=()

```
Contract_date & Contract_date::operator= (
            const Contract_date & contractDate_in )
```

= operátor az = jel helyes működéséhez

Parameters

contractDate_in	Bejövő contractDate

Returns

Visszaadandó contractDate

Definition at line 97 of file date.cpp.

4.4.3.5 setBegin() [1/2]

```
void Contract_date::setBegin (
            Date beg_in )
```

Beállítja a kezdeti időpontot Date-ből.

Parameters

beg_in	Bejövő dátum

Definition at line 71 of file date.cpp.

4.4.3.6 setBegin() [2/2]

```
void Contract_date::setBegin (
            int year,
            int month,
            int day )
```

Beállítja a kezdeti időpontot intekből.

Parameters

year	Beállítandó év
month	Beállítandó hónap
day	Beállítandó nap

Definition at line 76 of file date.cpp.

4.4.3.7 setEnd() [1/2]

```
void Contract_date::setEnd (
            Date end_in )
```

Beállítja a végső időpontot Date-ből.

Parameters

beg↩ _in	Bejövő dátum

Definition at line 82 of file date.cpp.

4.4.3.8 setEnd() [2/2]

```
void Contract_date::setEnd (
            int year,
            int month,
            int day )
```

Beállítja a végső időpontot intekből.

Parameters

year	Beállítandó év
month	Beállítandó hónap
day	Beállítandó nap

Definition at line 87 of file date.cpp.

The documentation for this class was generated from the following files:

- date.h
- date.cpp

4.5 Date Class Reference

Egy dátumot tárol time_point formátumba.

`#include <date.h>`

Public Member Functions

- Date (int year, int month, int day)
- std::chrono::system_clock::time_point into_date (int year, int month, int day)

  Időponttá alakít egy dátumot.
- void print_date () const

  Kiírja az időpontot.
- bool operator== (const Date &in) const

  == operátor összehasonlításhoz
- bool operator< (const Date &in) const

  < operátor összehasonlításhoz
- bool operator> (const Date &in) const

  > operátor összehasonlításhoz
- bool operator<= (const Date &in) const

  <= operátor összehasonlításhoz
- bool operator>= (const Date &in) const

  >= operátor összehasonlításhoz
- std::string toString () const

  Visszaadja a dátumot string formátumban.

4.5.1 Detailed Description

Egy dátumot tárol time_point formátumba.

Parameters

tp	tárolt dátum

Definition at line 16 of file date.h.

4.5.2 Constructor & Destructor Documentation

4.5.2.1 Date() [1/2]

```
Date::Date ( )  [inline]
```

Definition at line 21 of file date.h.

4.5.2.2 Date() [2/2]

```
Date::Date (
            int year,
            int month,
            int day )  [inline]
```

@Konstruktor

Parameters

year	Beállítandó év
month	Beállítandó hónap
day	Beállítandó nap

Definition at line 29 of file date.h.

4.5.3 Member Function Documentation

4.5.3.1 into_date()

```
std::chrono::system_clock::time_point Date::into_date (
            int year,
            int month,
            int day )
```

Időponttá alakít egy dátumot.

Parameters

year	Bejövő év
month	Bejövő hónap
day	Bejövő nap

Returns

időponttá alakított dátum

Definition at line 7 of file date.cpp.

4.5.3.2 operator<()

```
bool Date::operator< (
            const Date & in ) const
```

< operátor összehasonlításhoz

Parameters

in	Összehasonlítandó dátum

Returns

Kisebb?

Definition at line 28 of file date.cpp.

4.5.3.3 operator<=()

```
bool Date::operator<= (
            const Date & in ) const
```

<= operátor összehasonlításhoz

Parameters

in	Összehasonlítandó dátum

Returns

Kisebb vagy egyenlő?

Definition at line 36 of file date.cpp.

4.5.3.4 operator==()

```
bool Date::operator== (
            const Date & in ) const
```

== operátor összehasonlításhoz

Parameters

in	Összehasonlítandó dátum

Returns

Egyenlőek

Definition at line 24 of file date.cpp.

4.5.3.5 operator>()

```
bool Date::operator> (
            const Date & in ) const
```

> operátor összehasonlításhoz

Parameters

in	Összehasonlítandó dátum

Returns

Nagyobb?

Definition at line 32 of file date.cpp.

4.5.3.6 operator>=()

```
bool Date::operator>= (
            const Date & in ) const
```

>= operátor összehasonlításhoz

Parameters

in	Összehasonlítandó dátum

Returns

Nagyobb vagy egyenlő?

Definition at line 40 of file date.cpp.

4.5.3.7 print_date()

```
void Date::print_date ( ) const
```

Kiírja az időpontot.

Definition at line 17 of file date.cpp.

4.5.3.8 toString()

```
std::string Date::toString ( ) const
```

Visszaadja a dátumot string formátumban.

Returns

Definition at line 44 of file date.cpp.

The documentation for this class was generated from the following files:

- date.h
- date.cpp

4.6 MVM Class Reference

Ügyfeleket tároló class.

```
#include <MVM.h>
```

Public Member Functions

- MVM ()
  paraméter nélkül hívható konstruktor
- void addClient (Client *client_in)
  Hozzáad egy új klienst.
- void addContract (Client *client_in, ContractType ctype_in, int year_begin, int month_begin, int day_begin, int year_end, int month_end, int day_end, int invo_in_year, int invo_in_month, int invo_in_day, double tariff↩ _in=0.0, double balance_in=0.0, double consumption_in=0.0)
- Client * getClient (const std::string name_in, const std::string somekindofID)
  Visszaadja a keresett ügyfelet a neve alapján.
- const std::string getCompanyDetails ()
  Az MVM adatait adja vissza.
- void deleteClient (const std::string name_in, const std::string somekindofID)
  Kitöröl egy ügyfelet a neve alapján.
- void save ()
  Biztonsági mentés fájlba
- void load_from_save (int version_id_in)
  Beolvassa a ügyfeleket és a szerződéseiket egy fájlból.
- void load_from_console ()
  Belvas egy contractet a console-ról.
- ~MVM ()
  Destruktor.

Static Public Attributes

- static Company MVM_company = Company("Meseorszagi Villamos Muvek ZRT.", "10760798244")
- static int save_version = 0

4.6.1 Detailed Description

Ügyfeleket tároló class.

Parameters

clients	Ügyfeleket tároló indexelhető MyVektor

Definition at line 23 of file MVM.h.

4.6.2 Constructor & Destructor Documentation

4.6.2.1 MVM()

```
MVM::MVM ( )  [inline]
```

paraméter nélkül hívható konstruktor

Definition at line 73 of file MVM.h.

4.6.2.2 ~MVM()

```
MVM::~MVM ( )
```

Destruktor.

Definition at line 281 of file MVM.cpp.

4.6.3 Member Function Documentation

4.6.3.1 addClient()

```
void MVM::addClient (
            Client * client_in )
```

Hozzáad egy új klienst.

Eltárolja a klienst, amit előre létre kell hozni, de a felszabadítást az MVM osztály végzi

Parameters

client_in	Szerződést kötő ügyfél

Definition at line 10 of file MVM.cpp.

4.6.3.2 addContract()

```
void MVM::addContract (
            Client * client_in,
            ContractType ctype_in,
            int year_begin,
            int month_begin,
            int day_begin,
            int year_end,
            int month_end,
            int day_end,
            int invo_in_year,
            int invo_in_month,
            int invo_in_day,
            double tariff_in = 0.0,
            double balance_in = 0.0,
            double consumption_in = 0.0 )
```

@briefHozzáad egy szerződést

Parameters

client_in	Szerződést kötő ügyfél
ctype_in	Szerződés típusa
year_begin	Szerződéskötés kezdeti éve
month_begin	Szerződéskötés kezdeti hónapja
day_begin	Szerződéskötés kezdeti napja
year_end	Szerződéskötés végső éve
month_end	Szerződéskötés végső hónapja
day_end	Szerződéskötés végső napja
invo_in_year	Legutóbbi számlázás éve
invo_in_month	Legutóbbi számlázás hónapja
invo_in_day	Legutóbbi számlázás napja
tariff_in	KWh-kénti ár
balance_in	Bejövő egyenleg
consumption_in	Bejövő fogyasztás

Definition at line 14 of file MVM.cpp.

4.6.3.3 deleteClient()

```
void MVM::deleteClient (
            const std::string name_in,
            const std::string somekindofID )
```

Kitöröl egy ügyfelet a neve alapján.

Parameters

name_in	Törlendő ügyfél neve

Definition at line 34 of file MVM.cpp.

4.6.3.4 getClient()

```
Client * MVM::getClient (
            const std::string name_in,
            const std::string somekindofID )
```

Visszaadja a keresett ügyfelet a neve alapján.

A függvény mivel nem tudja hogy a client melyik alosztályát próbáljuk megkapni, így a módszer az hogy a subli-entinfóját hasonlítjuk össze azzal amire számítanánk hogy lesz

Parameters

name_in	Keresett ügyfél neve
somekindofID	A keresett ügyfél valamilyen azonosítója

Returns

Keresett ügyfél

Definition at line 19 of file MVM.cpp.

4.6.3.5 getCompanyDetails()

```
const std::string MVM::getCompanyDetails ( )
```

Az MVM adatait adja vissza.

Returns

MVM adatai

Definition at line 30 of file MVM.cpp.

4.6.3.6 load_from_console()

```
void MVM::load_from_console ( )
```

Belvas egy contractet a console-ról.

Definition at line 260 of file MVM.cpp.

4.6.3.7 load_from_save()

```
void MVM::load_from_save (
            int version_id_in )
```

Beolvassa a ügyfeleket és a szerződéseiket egy fájlból.

Parameters

beolvasandó	verzió azonosítója

Definition at line 238 of file MVM.cpp.

4.6.3.8 save()

```
void MVM::save ( )
```

Biztonsági mentés fájlba

Definition at line 38 of file MVM.cpp.

4.6.4 Member Data Documentation

4.6.4.1 MVM_company

```
Company MVM::MVM_company = Company("Meseorszagi Villamos Muvek ZRT.", "10760798244")  [static]
```

Az MVM adatai

Definition at line 63 of file MVM.h.

4.6.4.2 save_version

```
int MVM::save_version = 0  [static]
```

A mentések verziószámát tárolja.

Definition at line 68 of file MVM.h.

The documentation for this class was generated from the following files:

- MVM.h
- MVM.cpp

4.7 MyVector< T > Class Template Reference

Indexelhetőséghez std::vektor mintájára saját vektor template.

```
#include <myvektor.hpp>
```

Public Member Functions

- MyVector ()
  Alapértékes konstruktor.
- void push_back (T value)
  Bejövő objektum tárolása.
- size_t size () const
  Visszaadja a tárolt objektumok számát.
- size_t capacity () const
  Visszaadja a jelenlegi kapacitást.
- void erase (size_t pos)
  Egy elem eltávolítása a tárolt objektumok közül egy bizonyos helyről.
- void pop_back ()
  Utolsó objektum eltávolítása.
- T & operator[ ] (size_t i)
  [] operátor az indexelhetőséghez
- const T & operator[ ] (size_t i) const
  [] operátor az indexelhetőséghez constans formában is
- ~MyVector ()
  Destruktor.

4.7.1 Detailed Description

template<typename T>
class MyVector< T >

Indexelhetőséghez std::vektor mintájára saját vektor template.

Template Parameters

T	Tárolt objektumok fajtája

Parameters

m_buffer	Az adat buffer elejére mutató pointer
m_capacity	A maximális kapacitás a jelenlegi foglalt területtel
m_size	A jelenleg tárolt objektumok száma

Definition at line 21 of file myvektor.hpp.

4.7.2 Constructor & Destructor Documentation

4.7.2.1 MyVector()

```
template<typename T >
MyVector< T >::MyVector ( )  [inline]
```

Alapértékes konstruktor.

Definition at line 49 of file myvektor.hpp.

4.7.2.2 ~MyVector()

```
template<typename T >
MyVector< T >::~MyVector ( )  [inline]
```

Destruktor.

Definition at line 127 of file myvektor.hpp.

4.7.3 Member Function Documentation

4.7.3.1 capacity()

```
template<typename T >
size_t MyVector< T >::capacity ( ) const  [inline]
```

Visszaadja a jelenlegi kapacitást.

Returns

jelenlegi kapacitás

Definition at line 77 of file myvektor.hpp.

4.7.3.2 erase()

```
template<typename T >
void MyVector< T >::erase (
            size_t pos )  [inline]
```

Egy elem eltávolítása a tárolt objektumok közül egy bizonyos helyről.

Parameters

pos	Az eltávolítandó objektum indexe

Definition at line 85 of file myvektor.hpp.

4.7.3.3 operator[]() [1/2]

```
template<typename T >
T & MyVector< T >::operator[] (
            size_t i )  [inline]
```

[] operátor az indexelhetőséghez

Parameters

i	Az elérni kívánt objektum indexe

Returns

Visszaadandó objektum

Definition at line 109 of file myvektor.hpp.

4.7.3.4 operator[]() [2/2]

```
template<typename T >
const T & MyVector< T >::operator[] (
            size_t i ) const  [inline]
```

[] operátor az indexelhetőséghez constans formában is

Parameters

i	Az elérni kívánt objektum indexe

Returns

Visszaadandó objektum

Definition at line 119 of file myvektor.hpp.

4.7.3.5 pop_back()

```
template<typename T >
void MyVector< T >::pop_back ( )  [inline]
```

Utolsó objektum eltávolítása.

Definition at line 98 of file myvektor.hpp.

4.7.3.6 push_back()

```
template<typename T >
void MyVector< T >::push_back (
            T value )  [inline]
```

Bejövő objektum tárolása.

Parameters

value	Bejövő objektum

Definition at line 57 of file myvektor.hpp.

4.7.3.7 size()

```
template<typename T >
```

```
size_t MyVector< T >::size ( ) const  [inline]
```

Visszaadja a tárolt objektumok számát.

Returns

tárolt objektumok száma

Definition at line 69 of file myvektor.hpp.

The documentation for this class was generated from the following file:

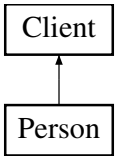
- myvektor.hpp

4.8 Person Class Reference

Magánszemély adatait tárolja.

```
#include <clients.h>
```

Inheritance diagram for Person:



Public Member Functions

- Person (const std::string name_in="", std::string ID_in="")

  Alapértékes konstruktor.
- void setID (std::string ID_in)

  Személyigazolvány beállítása.
- std::string getID () const

  Magánszemély személyigazolványszámát adja vissza.
- std::string getsub_Client_info ()

  Visszaadja a gyermek osztály személyes adatait kiíráshoz egy string-ben.
- ~Person ()

  Destruktor.

Public Member Functions inherited from Client

- Client (const std::string name_in)

  Alapértékes kontruktor.
- size_t getSumOfAll_Contract ()

  Visszaadja a kliens által eddig kötött szerződések számát.
- std::string getName () const

  Visszadja a kliens nevét.
- void addContract (Contract &contract_in)

  Szerződés hozzáadása a mycontracts hez.
- virtual std::string getsub_Client_info ()

  Visszaadja a gyermek osztály személyes adatait kiíráshoz egy string-ben.
- void save_invoice (size_t contract_index, const Date &date)

  Számlát állít ki fájlba.
- Contract & getContract (size_t contract_ID)

  Visszaadja azt a szerződést aminek az azonosítóját megadtuk.
- virtual ~Client ()

  Virtuális destruktor.

4.8.1 Detailed Description

Magánszemély adatait tárolja.

Parameters

ID	A személy személyazonosító száma 8 jegyű

Definition at line 83 of file clients.h.

4.8.2 Constructor & Destructor Documentation

4.8.2.1 Person()

```
Person::Person (
            const std::string name_in = "",
            std::string ID_in = "" )  [inline]
```

Alapértékes konstruktor.

Parameters

name_in	Bejövő magánszemély neve
ID_in	Bejövő magánszemély személyigazolványszáma
SumOfAll_contract	A kliens által eddig kötött szerződések száma (nem jelenlegi)

Definition at line 92 of file clients.h.

4.8.2.2 ~Person()

```
Person::~Person ( )
```

Destruktor.

Definition at line 48 of file clients.cpp.

4.8.3 Member Function Documentation

4.8.3.1 getID()

```
std::string Person::getID ( ) const
```

Magánszemély személyigazolványszámát adja vissza.

Returns

Magánszemély személyigazolványszáma

Definition at line 40 of file clients.cpp.

4.8.3.2 getsub_Client_info()

```
std::string Person::getsub_Client_info ( )  [virtual]
```

Visszaadja a gyermek osztály személyes adatait kiíráshoz egy string-ben.

Returns

gyermek osztály személyes adatai kiíráshoz egy string-ben

Reimplemented from Client.

Definition at line 36 of file clients.cpp.

4.8.3.3 setID()

```
void Person::setID (
            std::string ID_in )
```

Személyigazolvány beállítása.

Parameters

ID↩ _in	Beérkező személyigazolványszám

Definition at line 44 of file clients.cpp.

The documentation for this class was generated from the following files:

- clients.h
- clients.cpp

Chapter 5

File Documentation

5.1 clients.cpp

```
00001 //
00002 // Created by Jajecnik Marcell on 2023. 05. 04..
00003 //
00004
00005 #include "clients.h"
00006
00007 std::string Client::getName() const{
00008     return name;
00009 }
00010
00011 void Client::addContract(Contract& contract_in) {
00012     if(contract_in.getContractID()>SumOfAll_contract) SumOfAll_contract = contract_in.getContractID();
00013     mycontracts.push_back(contract_in);
00014 }
00015
00016 size_t Client::getSumOfAll_Contract() {
00017     return SumOfAll_contract;
00018 }
00019
00020 std::string Client::getsub_Client_info() {
00021     return "Client name: " + getName();
00022 }
00023
00024 void Client::save_invoice(size_t contract_index, const Date& date) {
00025     getContract(contract_index).invoice(date);
00026 }
00027
00028 Contract &Client::getContract(size_t contract_index) {
00029     if (contract_index<=0 || contract_index>getSumOfAll_Contract()) throw std::range_error("Túl/alul
      indexelés"); // Programozói hiba
00030     for (size_t i = 0; i < mycontracts.size(); ++i) {
00031         if (mycontracts[i].getContractID()==contract_index) return mycontracts[i];
00032     }
00033     throw std::range_error("Már törölt szerződés"); // Lehetne saját exception classal
00034 }
00035
00036 std::string Person::getsub_Client_info() {
00037     return "Person name: " + getName() + ", ID: " + ID;
00038 }
00039
00040 std::string Person::getID() const {
00041     return ID;
00042 }
00043
00044 void Person::setID(std::string ID_in) {
00045     ID=ID_in;
00046 }
00047
00048 Person::~Person() {
00049
00050 }
00051
00052 std::string Company::getID() const {
00053     return Tax_ID;
00054 }
00055
00056 void Company::setID(std::string ID_in) {
00057     Tax_ID=ID_in;
```

```
00058 }
00059
00060 std::string Company::getsub_Client_info() {
00061     return "Company name: " + getName() + ", Tax ID: " + Tax_ID;
00062 }
00063
00064 Company::~Company() {
00065
00066 }
```

5.2 clients.h

```
00001 //
00002 // Created by Jajecnik Marcell on 2023. 05. 04..
00003 //
00004
00005 #ifndef SKELETON_CLIENTS_H
00006 #define SKELETON_CLIENTS_H
00007
00008 #include "date.h"
00009 #include "myvektor.hpp"
00010 #include "contracts.h"
00011 #include <string>
00012
00014 class Contract;
00015
00021 class Client{
00022 private:
00023     std::string name;
00024     MyVector<Contract> mycontracts;
00025     size_t SumOfAll_contract;
00026
00027 public:
00033     Client(const std::string name_in):name(std::string(name_in)), SumOfAll_contract(0){}
00034
00039     size_t getSumOfAll_Contract();
00040
00045     std::string getName() const;
00046
00051     void addContract(Contract& contract_in);
00052
00058     virtual std::string getsub_Client_info();
00059
00065     void save_invoice(size_t contract_index, const Date& date);
00066
00072     Contract& getContract(size_t contract_ID);
00073
00077     virtual ~Client(){};
00078 };
00079
00083 class Person: public Client{
00084 private:
00085     std::string ID;
00086 public:
00092     Person(const std::string name_in="", std::string ID_in="")
00093     : Client(std::string(name_in)), ID(ID_in)
00094     {}
00095
00100     void setID(std::string ID_in);
00101
00106     std::string getID() const;
00107
00113     std::string getsub_Client_info();
00114
00118     ~Person();
00119 };
00120
00124 class Company: public Client{
00125 private:
00126     std::string Tax_ID;
00127 public:
00128
00135     Company(std::string name_in="", std::string ID_in="")
00136     : Client(name_in), Tax_ID(ID_in)
00137     {}
00138
00143     void setID(std::string ID_in);
00144
00149     std::string getID() const;
00150
00156     std::string getsub_Client_info();
00157
00161     ~Company();
```

```
00162 };
00163
00164 #endif //SKELETON_CLIENTS_H
```

5.3 contracts.cpp

```
00001 //
00002 // Created by Jajecnik Marcell on 2023. 05. 05..
00003 //
00004
00005 #include "contracts.h"
00006
00007 void Contract::setCtype(ContractType ctype_in) {
00008     contract_type = ctype_in;
00009 }
00010
00011 void Contract::setTariff(double tariff_in) {
00012     tariff = tariff_in;
00013 }
00014
00015 void Contract::setBalance(double balance_in) {
00016     balance = balance_in;
00017 }
00018
00019 void Contract::addConsumption(double con_in) {
00020     consumption += con_in;
00021 }
00022
00023 void Contract::setLast_invoicing(Date last_in) {
00024     last_invoicing = last_in;
00025 }
00026
00027 void Contract::setCtime(Contract_date ctime_in) {
00028     contract_time = ctime_in;
00029 }
00030
00031 Client* Contract::getClient() {
00032     return client;
00033 }
00034
00035 size_t Contract::getContractID() const{
00036     return contract_ID;
00037 }
00038
00039 ContractType Contract::getCtype() const {
00040     return contract_type;
00041 }
00042
00043 double Contract::getTariff() const {
00044     return tariff;
00045 }
00046
00047 double Contract::getBalance() const {
00048     return balance;
00049 }
00050
00051 double Contract::getConsumption() const {
00052     return consumption;
00053 }
00054
00055 Contract_date Contract::getCtime() const {
00056     return contract_time;
00057 }
00058
00059 const std::string Contract::Ctype_toString() const {
00060     switch (contract_type) {
00061     case ContractType::Regular: return "Regular";
00062     case ContractType::VIP: return "VIP";
00063     case ContractType::Premium: return "Premium";
00064     case ContractType::Corporate: return "Corporate";
00065     case ContractType::Student: return "Student";
00066     case ContractType::SeniorCitizen: return "SeniorCitizen";
00067     case ContractType::Government: return "Government";
00068     case ContractType::Onetime: return "Onetime";
00069     default: return "Unknown";
00070     }
00071 }
00072
00073 Date Contract::getLast_invoicing() const {
00074     return last_invoicing;
00075 }
00076
00077 void Contract::pay(double amount) {
```

```
00078     balance += amount;
00079 }
00080
00081 void Contract::invoice( const Date &today) {
00082     std::ofstream outfile;
00083     std::string filename = client->getName() + "_" + std::to_string(contract_ID) + ".txt";
00084
00085     outfile.open(filename);
00086     // Write contract details
00087     outfile « "Contract details:" « std::endl;
00088         outfile« "Seller details:" « std::endl;
00089     outfile«"Seller name: "« MVM::MVM_company.getsub_Client_info() « std::endl;
00090     outfile « "Client name: " « getClient()->getName() « std::endl;
00091     outfile « "Client ID: " « getClient()->getsub_Client_info() « std::endl;
00092     outfile « "Contract type: " « Ctype_toString() « std::endl;
00093     outfile « "Contract start date: " « getCtime().getBegin().toString() « std::endl;
00094     outfile « "Contract end date: " « getCtime().getEnd().toString() « std::endl;
00095     outfile « "Last invoicing date: " « getLast_invoicing().toString() « std::endl;
00096     outfile « "Tariff: " « std::fixed « std::setprecision(2) « getTariff() « " Ft/KWh" « std::endl;
00097     outfile « "Balance: " « std::fixed « std::setprecision(2) « getBalance() « " Ft" « std::endl;
00098     outfile « "Consumption since last invoicing: " « std::fixed « std::setprecision(2) «
      getConsumption() « " KWh" « std::endl;
00099
00100     // Calculate invoice details
00101     double consumption = getConsumption();
00102     double prev_balance =getBalance();
00103     double tariff = getTariff();
00104     double vat = 0.27; // Value added tax
00105     double subtotal = consumption * tariff;
00106     double total = subtotal * (1 + vat);
00107     double new_balance = prev_balance + total;
00108
00109     // Write invoice details
00110     outfile « std::endl;
00111     outfile « "Invoice details:" « std::endl;
00112     outfile « "Invoice date: " « today.toString() « std::endl;
00113     outfile « "Previous balance: " « std::fixed « std::setprecision(2) « prev_balance « " Ft" «
      std::endl;
00114     outfile « "Tariff: " « std::fixed « std::setprecision(2) « tariff « " Ft/KWh" « std::endl;
00115     outfile « "Consumption: " « std::fixed « std::setprecision(2) « consumption « " KWh" « std::endl;
00116     outfile « "Subtotal: " « std::fixed « std::setprecision(2) « subtotal « " Ft" « std::endl;
00117     outfile « "VAT (" « std::fixed « std::setprecision(2) « vat*100 « "%): " « std::fixed «
      std::setprecision(2) « subtotal * vat « " Ft" « std::endl;
00118     outfile « "Total: " « std::fixed « std::setprecision(2) « total « " Ft" « std::endl;
00119     outfile « "New balance: " « std::fixed « std::setprecision(2) « new_balance « " Ft" « std::endl;
00120
00121     outfile.close();
00122 }
00123
00124 Contract::~Contract() {
00125
00126 }
```

5.4 contracts.h

```
00001 //
00002 // Created by Jajecnik Marcell on 2023. 05. 05..
00003 //
00004
00005 #ifndef SKELETON_CONTRACTS_H
00006 #define SKELETON_CONTRACTS_H
00007 #include "clients.h"
00008 #include "iomanip"
00009 #include "fstream"
00010 #include "MVM.h"
00011 #include <string>
00012
00013 class Client;
00014
00018 enum class ContractType {
00019     Regular,
00020     VIP,
00021     Premium,
00022     Corporate,
00023     Student,
00024     SeniorCitizen,
00025     Government,
00026     Onetime
00027 };
00028
00041 class Contract {
00042
00043 public:
```

```
00044
00048     Contract() {};
00049
00068     Contract(Client* client_in, ContractType ctype_in, int year_begin, int month_begin, int day_begin,
      int year_end, int month_end, int day_end, int invo_in_year, int invo_in_month, int invo_in_day, size_t
      ContractID_in ,double tariff_in=0.0, double balance_in=0.0, double consumption_in=0.0)
00069             :client(client_in),contract_ID(ContractID_in), contract_type(ctype_in),
      contract_time(year_begin, month_begin, day_begin, year_end, month_end, day_end),
      last_invoicing(invo_in_year, invo_in_month, invo_in_day), tariff(tariff_in), balance(balance_in),
      consumption(consumption_in)
00070     {}
00071
00072     // Setter functions
00077     void setCtype(ContractType ctype_in);
00078
00083     void setTariff(double tariff_in);
00084
00089     void setBalance(double balance_in);
00090
00095     void addConsumption(double con_in);
00096
00101     void setLast_invoicing(Date last_in);
00102
00107     void setCtime(Contract_date ctime_in);
00108
00113     Client* getClient();
00114
00119     size_t getContractID() const;
00120
00125    ContractType getCtype() const;
00126
00131     double getTariff() const;
00132
00137     double getBalance() const;
00138
00143     double getConsumption() const;
00144
00149     Contract_date getCtime() const;
00150
00155     const std::string Ctype_toString() const;
00156
00161     Date getLast_invoicing() const;
00162
00167     void pay(double amount);
00168
00173     void invoice( const Date& today);
00174
00178     ~Contract();
00179 private:
00180
00181     Client* client;
00182     int contract_ID;
00183     ContractType contract_type;
00184     Contract_date contract_time;
00185     Date last_invoicing;
00186     double tariff;
00187     double balance;
00188     double consumption;
00189
00190 };
00191
00192 #endif //SKELETON_CONTRACTS_H
```

5.5 date.cpp

```
00001 //
00002 // Created by Jajecnik Marcell on 2023. 05. 04..
00003 //
00004
00005 #include "date.h"
00006
00007 std::chrono::system_clock::time_point Date::into_date(int year, int month, int day) {
00008     if (year < 1900 || month < 1 || month>12 || day < 1 || day>31)
00009     {
00010         throw std::range_error("Hibas datum, vagy 1900 elotti");
00011     }
00012     std::tm tm = { 0, 0, 0, day, month - 1, year - 1900 }; // Month is 0-based, year is 1900-based
00013     std::time_t tt = std::mktime(&tm);
00014     return std::chrono::system_clock::from_time_t(tt);
00015 }
00016
00017 void Date::print_date() const {
00018     std::time_t tt = std::chrono::system_clock::to_time_t(tp);
```

```
00019     char buffer[11];
00020     std::strftime(buffer, 11, "%Y-%m-%d", std::localtime(&tt));
00021     std::cout « "Date entered: " « buffer;
00022 }
00023
00024 bool Date::operator==(const Date& in) const {
00025     return  tp == in.tp;
00026 }
00027
00028 bool Date::operator<(const Date& in) const {
00029     return tp < in.tp;
00030 }
00031
00032 bool Date::operator>(const Date& in) const {
00033     return tp > in.tp;
00034 }
00035
00036 bool Date::operator<=(const Date& in) const {
00037     return tp <= in.tp;
00038 }
00039
00040 bool Date::operator>=(const Date& in) const {
00041     return tp >= in.tp;
00042 }
00043
00044 std::string Date::toString() const {
00045     // Convert the time_point to a time_t
00046     std::time_t tt = std::chrono::system_clock::to_time_t(tp);
00047
00048     // Convert the time_t to a struct tm
00049     std::tm tm = *std::localtime(&tt);
00050
00051     // Format the date as a string
00052     std::stringstream ss;
00053     ss « std::put_time(&tm, "%Y-%m-%d");
00054     return ss.str();
00055 }
00056
00057
00058 std::ostream& operator«(std::ostream os, const Date& date) {
00059     date.print_date();
00060     return os;
00061 }
00062
00063 Date Contract_date::getBegin() const {
00064     return begin;
00065 }
00066
00067 Date Contract_date::getEnd() const {
00068     return end;
00069 }
00070
00071 void Contract_date::setBegin(Date beg_in) {
00072     if (beg_in>end) throw std::range_error("A kezdeti időpont nem lehet nagyobb mint a végső dátum");
00073     begin=beg_in;
00074 }
00075
00076 void Contract_date::setBegin(int year, int month, int day) {
00077     Date new_begin(year, month, day);
00078     if (new_begin > end) throw std::range_error("A kezdeti időpont nem lehet nagyobb mint a végső
      dátum");
00079     begin = new_begin;
00080 }
00081
00082 void Contract_date::setEnd(Date end_in) {
00083     if (end_in < begin) throw std::range_error("A végső időpont nem lehet kisebb mint a kezdeti
      dátum");
00084     end= end_in;
00085 }
00086
00087 void Contract_date::setEnd(int year, int month, int day) {
00088     Date new_end(year, month, day);
00089     if (new_end < begin) throw std::range_error("A végső időpont nem lehet kisebb mint a kezdeti
      dátum");
00090     end = new_end;
00091 }
00092
00093 bool Contract_date::contains(const Date& date_in) const {
00094     return (date_in >= begin && date_in <= end);
00095 }
00096
00097 Contract_date& Contract_date::operator=(const Contract_date& contractDate_in) {
00098         this->begin = contractDate_in.begin;
00099         this->end = contractDate_in.end;
00100     return *this;
00101 }
00102
```

00103

5.6 date.h

```
00001 //
00002 // Created by Jajecnik Marcell on 2023. 05. 04..
00003 //
00004
00005 #ifndef SKELETON_DATE_H
00006 #define SKELETON_DATE_H
00007
00008 #include <iostream>
00009 #include <chrono>
00010 #include "sstream"
00011 #include "iomanip"
00012
00016 class Date {
00017 private:
00018     std::chrono::system_clock::time_point tp;
00019
00020 public:
00021     Date(){}
00022
00029     Date(int year, int month, int day): tp(into_date(year,month,day)) {}
00030
00038     std::chrono::system_clock::time_point into_date(int year, int month, int day);
00039
00042     void print_date() const;
00043
00049     bool operator==(const Date& in) const;
00050
00056     bool operator<(const Date& in) const;
00057
00063     bool operator>(const Date& in) const;
00064
00070     bool operator<=(const Date& in) const;
00071
00077     bool operator>=(const Date& in) const;
00078
00084     std::string toString() const;
00085 };
00086
00093 std::ostream& operator«(std::ostream os, const Date& date);
00094
00095
00096
00101 class Contract_date {
00102 private:
00103     Date begin;
00104     Date end;
00105
00106 public:
00107
00111     Contract_date(){}
00112
00119     Contract_date(Date begin_date, Date end_date):begin(begin_date),end(end_date) {
00120         if(begin_date > end_date) throw std::range_error("A szerződés kezdete később van mint a
      szerződés vége");
00121     }
00122
00132     Contract_date(int year_begin, int month_begin, int day_begin, int year_end, int month_end, int
      day_end)
00133     {
00134         Date start(year_begin, month_begin, day_begin);
00135         Date end(year_end, month_end, day_end);
00136         *this = Contract_date(start, end);
00137     }
00138
00143     Date getBegin() const;
00144
00149     Date getEnd() const;
00150
00155     void setBegin(Date beg_in);
00156
00163     void setBegin(int year, int month, int day);
00164
00169     void setEnd(Date end_in);
00170
00177     void setEnd(int year, int month, int day);
00178
00184     bool contains(const Date& date_in) const;
00185
00186
```

```
00192     Contract_date& operator=(const Contract_date& contractDate_in);
00193 };
00194
00195
00196 #endif //SKELETON_DATE_H
```

5.7 MVM.cpp

```
00001 //
00002 // Created by Jajecnik Marcell on 2023. 05. 04..
00003 //
00004
00005 #include "MVM.h"
00006
00007 Company MVM::MVM_company = Company("Meseorszagi Villamos Muvek ZRT.", "10760798244");
00008 int MVM::save_version = 0;
00009
00010 void MVM::addClient(Client* client_in) {
00011     clients.push_back(client_in);
00012 }
00013
00014 void MVM::addContract(Client* client_in, ContractType ctype_in, int year_begin, int month_begin, int
      day_begin, int year_end, int month_end, int day_end, int invo_in_year, int invo_in_month, int
      invo_in_day, double tariff_in, double balance_in, double consumption_in) {
00015     Contract new_contract(client_in, ctype_in, year_begin, month_begin, day_begin, year_end,
      month_end, day_end, invo_in_year, invo_in_month, invo_in_day, (client_in->getSumOfAll_Contract() + 1),
      tariff_in, balance_in, consumption_in);
00016     client_in->addContract(new_contract);
00017 }
00018
00019 Client *MVM::getClient(const std::string name_in, const std::string somekindofID) {
00020     for (size_t i = 0; i < clients.size(); ++i) {
00021         std::string subclient_info = clients[i]->getsub_Client_info();
00022         if (name_in == clients[i]->getName() &&
00023             ((subclient_info.substr(subclient_info.length() - 11, 11) == somekindofID ||
00024                 subclient_info.substr(subclient_info.length() - 8, 8) == somekindofID)))
00025         return clients[i];
00026     }
00027     return nullptr;
00028 }
00029
00030 const std::string MVM::getCompanyDetails() {
00031     return MVM_company.getsub_Client_info();
00032 }
00033
00034 void MVM::deleteClient(const std::string name_in, const std::string somekindofID) {
00035     delete getClient(name_in, somekindofID);
00036 }
00037
00038 void MVM::save() {
00039     save_version++;
00040     std::string filename = "save_" + std::to_string(save_version) + ".txt";
00041     std::ofstream outputFile;  // Open the file for writing
00042     outputFile.open(filename, std::ios::out);
00043
00044     if (!outputFile.is_open()) {
00045         std::cerr « "Error opening file for writing: " « filename « std::endl;
00046         throw std::ios_base::failure("Error opening file for writing: " + filename);
00047     }
00048     size_t client_number = clients.size();
00049     for (size_t i = 0; i < client_number; i++)
00050     {
00051
00052         size_t contract_number = clients[i]->getSumOfAll_Contract();
00053         for (size_t j = 1; j <= contract_number; j++)
00054         {
00055             Contract contract_out;
00056             try {
00057                 contract_out = clients[i]->getContract(j); // Throws exception when contract is
      invalid
00058             }
00059             catch (const std::range_error& e) {
00060                 continue;
00061             }
00062
00063             outputFile « contract_out.getClient()->getName() « std::endl;
00064             outputFile « contract_out.getClient()->getsub_Client_info() « std::endl;
00065             outputFile « contract_out.getContractID() « std::endl;
00066             outputFile « contract_out.Ctype_toString() « std::endl;
00067             outputFile « contract_out.getCtime().getBegin().toString() « std::endl;
00068             outputFile « contract_out.getCtime().getEnd().toString() « std::endl;
00069             outputFile « contract_out.getLast_invoicing().toString() « std::endl;
00070             outputFile « std::fixed « std::setprecision(2) « contract_out.getTariff()  « std::endl;
```

```
00071             outputFile « std::fixed « std::setprecision(2) « contract_out.getBalance() « std::endl;
00072             outputFile « std::fixed « std::setprecision(2) « contract_out.getConsumption()  «
      std::endl;
00073         }
00074
00075     }
00076     outputFile.close();
00077 }
00078
00079 bool MVM::is_all_number(std::string& str_in) {
00080     for (size_t i = 0; i < str_in.length(); i++)
00081     {
00082         if (!std::isdigit(str_in[i])) return false;
00083     }
00084     return true;
00085 }
00086
00087 bool MVM::is_string_double(std::string& str_in) {
00088     size_t str_len = str_in.length();
00089     int i = 0;
00090     while (i<str_len&&str_in[i]!='.')
00091     {
00092         if (!std::isdigit(str_in[i])) return false;
00093         i++;
00094     }
00095     i++;
00096     while (i < str_len)
00097     {
00098         if (!std::isdigit(str_in[i])) return false;
00099         i++;
00100     };
00101
00102     return true;
00103 }
00104
00105 ContractType MVM::Ctype_fromString(const std::string contract_type) {
00106     if (contract_type == std::string("Regular"))             return ContractType::Regular;
00107     if (contract_type == std::string("VIP"))                 return ContractType::VIP;
00108     if (contract_type == std::string("Premium"))             return ContractType::Premium;
00109     if (contract_type == std::string("Corporate"))           return ContractType::Corporate;
00110     if (contract_type == std::string("Student"))             return ContractType::Student;
00111     if (contract_type == std::string("SeniorCitizen"))       return ContractType::SeniorCitizen;
00112     if (contract_type == std::string("Government"))          return ContractType::Government;
00113     if (contract_type == std::string("Onetime"))             return ContractType::Onetime;
00114     throw std::invalid_argument("Nem letezo szerzodes tipus");
00115     return ContractType::Onetime;
00116 }
00117
00118 void MVM::load_from_stream(std::istream& is) {
00119     std::string name_in, sub_client_info_in, contract_ID_in, ctype_in, begin, end_in,
      last_invoicing_in, tariff_in, balance_in, consumption_in;
00120     if (
00121         !std::getline(is, name_in)) throw std::out_of_range("End of file");
00122     std::getline(is, sub_client_info_in);
00123     if (&is != &std::cin) std::getline(is, contract_ID_in);
00124     std::getline(is, ctype_in);
00125     std::getline(is, begin);
00126     std::getline(is, end_in);
00127     std::getline(is, last_invoicing_in);
00128     std::getline(is, tariff_in);
00129     std::getline(is, balance_in);
00130     std::getline(is, consumption_in);
00131
00132     Client* client_in;
00133     if (sub_client_info_in.substr(0, 6) == "Person") {
00134         client_in = getClient(name_in, sub_client_info_in.substr(sub_client_info_in.length() - 8, 8));
00135         if (client_in == nullptr) {
00136             client_in = new Person(name_in, sub_client_info_in.substr(sub_client_info_in.length() - 8,
      8));
00137             addClient(client_in);
00138         }
00139     }
00140     else if(sub_client_info_in.substr(0, 7) == "Company"){
00141         client_in = getClient(name_in, sub_client_info_in.substr(sub_client_info_in.length() - 11,
      11));
00142         if (client_in == nullptr) {
00143             client_in = new Company(name_in, sub_client_info_in.substr(sub_client_info_in.length() -
      11, 11));
00144             addClient(client_in);
00145         }
00146     }
00147     else
00148     {
00149         throw std::invalid_argument("Rossz bejovo adat");
00150     }
00151
00152     if (
```

```
00153         begin.length() != 10  ||
00154         end_in.length() != 10 ||
00155         last_invoicing_in.length() != 10 ||
00156         !is_all_number(begin.substr(0, 4)) ||
00157         !is_all_number(begin.substr(5, 2)) ||
00158         !is_all_number(begin.substr(8, 2)) ||
00159         !is_all_number(end_in.substr(0, 4)) ||
00160         !is_all_number(end_in.substr(5, 2)) ||
00161         !is_all_number(end_in.substr(8, 2)) ||
00162         !is_all_number(last_invoicing_in.substr(0, 4)) ||
00163         !is_all_number(last_invoicing_in.substr(5, 2)) ||
00164         !is_all_number(last_invoicing_in.substr(8, 2)) ||
00165         !is_string_double(tariff_in) ||
00166         !is_string_double(balance_in) ||
00167         !is_string_double(consumption_in)
00168         )
00169         throw std::invalid_argument("Rossz bejovo adat");
00170
00171     //Conversion
00172     int year_begin, month_begin, day_begin, year_end, month_end, day_end, invo_in_year, invo_in_month,
      invo_in_day;
00173
00174     year_begin = std::stoi(begin.substr(0, 4));
00175     month_begin = std::stoi(begin.substr(5, 2));
00176     day_begin = std::stoi(begin.substr(8, 2));
00177     year_end = std::stoi(end_in.substr(0, 4));
00178     month_end = std::stoi(end_in.substr(5, 2));
00179     day_end = std::stoi(end_in.substr(8, 2));
00180     invo_in_year = std::stoi(last_invoicing_in.substr(0, 4));
00181     invo_in_month = std::stoi(last_invoicing_in.substr(5, 2));
00182     invo_in_day = std::stoi(last_invoicing_in.substr(8, 2));
00183
00184
00185
00186
00187     ContractType contract_type_in = Ctype_fromString(ctype_in);
00188
00189     double tariffin, balancein, consumptionin;
00190     tariffin = std::stod(tariff_in);
00191     balancein = std::stod(balance_in);
00192     consumptionin = std::stod(consumption_in);
00193
00194     if (&is != &std::cin) {
00195         size_t contractIDin = stoi(contract_ID_in);
00196         Contract contract_in(
00197             client_in,
00198             contract_type_in,
00199             year_begin,
00200             month_begin,
00201             day_begin,
00202             year_end,
00203             month_end,
00204             day_end,
00205             invo_in_year,
00206             invo_in_month,
00207             invo_in_day,
00208             contractIDin,
00209             tariffin,
00210             balancein,
00211             consumptionin
00212         );
00213         client_in->addContract(contract_in);
00214
00215     }
00216     else
00217     {
00218         addContract(client_in,
00219             contract_type_in,
00220             year_begin,
00221             month_begin,
00222             day_begin,
00223             year_end,
00224             month_end,
00225             day_end,
00226             invo_in_year,
00227             invo_in_month,
00228             invo_in_day,
00229             tariffin,
00230             balancein,
00231             consumptionin);
00232     }
00233
00234
00235
00236 };
00237
00238 void MVM::load_from_save(int version_id_in) {
```

```
00239     std::string filename = "save_" + std::to_string(version_id_in) + ".txt";
00240     std::ifstream inputFile;
00241     inputFile.open(filename, std::ios::in);
00242     if (!inputFile.is_open()) {
00243         std::cerr « "Error opening file for reading: " « filename « std::endl;
00244         throw std::ios_base::failure("Error opening file for reading: " + filename);
00245     }
00246     while (inputFile)
00247     {
00248         try {
00249             load_from_stream(inputFile);
00250         }
00251         catch (const std::invalid_argument& ia) {
00252             std::cerr « "Invalid argument: " « ia.what() « '\n';
00253             throw std::invalid_argument("Rossz forrasfajl");
00254         }
00255         catch (const std::out_of_range& end) { break; }
00256     }
00257     inputFile.close();
00258 }
00259
00260 void MVM::load_from_console() {
00261     std::cout « "Kerem adja meg a szerzodés adatait az alabbi minta alapjan:" « std::endl;
00262     std::cout«"Nev" « std::endl;
00263     std::cout« "Ugyfel tipusa es azonositoja (Person/Company, ceg eseten az adoazonosito 11 jegyu,
    maganszemely szemelyigazolvanyszama 8 kerekteru)" « std::endl;
00264     std::cout« "Szerzodes tipusa:
    (Regular/VIP/Premium/Corporate/Student/SeniorCitizen/Government/Onetime)" « std::endl;
00265     std::cout«"Szerzodes kezdete: (pl.: 2004-03-11)" « std::endl;
00266     std::cout« "Szerzodes vege: (pl.: 2004-03-11)" « std::endl;
00267     std::cout«"Legutobbi szamlazas napja: (pl.: 2004-03-11) ha meg nem volt a szerzodes kezdetének
    napja" « std::endl;
00268     std::cout« "Tarifa erteke (kw/h-ban megadva, mertekegyseg nelkul)" « std::endl;
00269     std::cout«"Egyenleg erteke (Ft-ban megadva, mertekegyseg nelkul)" « std::endl;
00270     std::cout«"Fogyasztas erteke (kw/h-ban megadva, mertekegyseg nelkul)" « std::endl;
00271     try {
00272         load_from_stream(std::cin);
00273     }
00274     catch (const std::invalid_argument& ia) {
00275         std::cerr « "Invalid argument: " « ia.what() « '\n';
00276             throw std::invalid_argument("Rosszul megadott adat(ok)");
00277     }
00278     std::cout « "Bevitel sikeres!" « std::endl;
00279 }
00280
00281 MVM::~MVM(){
00282     for (size_t i = 0; i < clients.size(); i++)
00283     {
00284         delete clients[i];
00285     }
00286 }
```

5.8 MVM.h

```
00001 //
00002 // Created by Jajecnik Marcell on 2023. 05. 04..
00003 //
00004
00005 #ifndef SKELETON_MVM_H
00006 #define SKELETON_MVM_H
00007
00008 #include "clients.h"
00009 #include "myvektor.hpp"
00010 #include "date.h"
00011 #include "contracts.h"
00012 #include <iostream>
00013 #include <chrono>
00014
00015 class Client;
00016 class Company;
00017 enum class ContractType;
00018
00023 class MVM {
00024 private:
00028     MyVector<Client *> clients;
00029
00036     ContractType Ctype_fromString(const std::string contract_type);
00037
00041     void load_from_stream(std::istream& is);
00042
00049     bool is_all_number(std::string& str_in);
00050
00057     bool is_string_double(std::string& str_in);
```

```
00058
00059 public:
00063     static Company MVM_company;
00064
00068     static int save_version;
00069
00073     MVM() {}
00074
00079     void addClient(Client* client_in);
00080
00098     void addContract(Client* client_in, ContractType ctype_in, int year_begin, int month_begin, int
      day_begin, int year_end, int month_end, int day_end, int invo_in_year, int invo_in_month, int
      invo_in_day, double tariff_in = 0.0, double balance_in = 0.0, double consumption_in = 0.0);
00099
00106     Client* getClient(const std::string name_in,const std::string somekindofID);
00107
00112     const std::string getCompanyDetails();
00113
00118     void deleteClient(const std::string name_in, const std::string somekindofID);
00119
00121     void save();
00122
00123
00128     void load_from_save(int version_id_in);
00129
00131     void load_from_console();
00132
00136     ~MVM();
00137 };
00138
00139 #endif //SKELETON_MVM_H
```

5.9 myvektor.hpp

```
00001 //
00002 // Created by Jajecnik Marcell on 2023. 05. 04..
00003 //
00004
00005 #ifndef SKELETON_MYVEKTOR_HPP
00006 #define SKELETON_MYVEKTOR_HPP
00007
00008 #include <iostream>
00009 #include <cstring>
00010 #include <stdexcept>
00011
00020 template<typename T>
00021 class MyVector {
00022 private:
00023
00024     T* m_buffer;
00025
00026     size_t m_capacity;
00027
00028     size_t m_size;
00029
00034     void reserve(size_t new_capacity) {
00035         if (new_capacity <= m_capacity) {
00036             return;
00037         }
00038         T* new_buffer = new T[new_capacity];
00039         memcpy(new_buffer, m_buffer, m_size * sizeof(T));
00040         delete[] m_buffer;
00041         m_buffer = new_buffer;
00042         m_capacity = new_capacity;
00043     }
00044
00045 public:
00049     MyVector(): m_capacity(10), m_size(0) {
00050         m_buffer = new T[m_capacity];
00051     }
00052
00057     void push_back(T value) {
00058         if (m_size == m_capacity) {
00059             reserve(m_capacity * 2);
00060         }
00061         m_buffer[m_size] = value;
00062         m_size++;
00063     }
00064
00069     size_t size() const {
00070         return m_size;
00071     }
00072
```

```
00077     size_t capacity() const {
00078         return m_capacity;
00079     }
00080
00085     void erase(size_t pos) {
00086         if (pos >= m_size) {
00087             return;
00088         }
00089         for (size_t i = pos; i < m_size - 1; i++) {
00090             m_buffer[i] = m_buffer[i + 1];
00091         }
00092         m_size--;
00093     }
00094
00098     void pop_back() {
00099         if (m_size > 0) {
00100             m_size--;
00101         }
00102     }
00103
00109     T& operator[](size_t i) {
00110         if ((i<0 || i>=m_size)) throw std::range_error("Index error");
00111         return m_buffer[i];
00112     }
00113
00119     const T& operator[](size_t i) const {
00120         if ((i<0 || i>=m_size)) throw std::range_error("Index error");
00121         return m_buffer[i];
00122     };
00123
00127     ~MyVector() {
00128         delete[] m_buffer;
00129     }
00130 };
00131
00132
00133 #endif //SKELETON_MYVEKTOR_HPP
```

5.10 Test.cpp

```
00001 #include <iostream>
00002
00003 #include "gtest_lite.h"
00004 #include "memtrace.h"
00005 #include "MVM.h"
00006
00018 #define TST 9
00019
00020
00021
00022 #define MEMTRACE_H
00023
00024 int main() {
00025
00026 #if TST > 0
00027
00028     TEST(MyVector, tesztek) {
00029
00030         MyVector<int> int_test;
00031         MyVector<double> double_test;
00032         EXPECT_EQ(10, int_test.capacity()); // Alapertelmezett ertek
00033         EXPECT_EQ(0, int_test.size());
00034         int_test.push_back(1);
00035         EXPECT_EQ(1, int_test.size());
00036         int_test.pop_back();
00037         EXPECT_EQ(0, int_test.size());
00038         int_test.pop_back(); //Torles ha nincs semmi bajt okoz-e?
00039
00040         for(size_t i = 0; i < 11; ++i) {
00041             int_test.push_back(i);
00042             double_test.push_back(i);
00043         }
00044         EXPECT_EQ(20, int_test.capacity());
00045         EXPECT_EQ(11, int_test.size());
00046
00047         for(size_t i = 0; i < 11; ++i) {
00048             int_test.erase(0);
00049             double_test.pop_back();
00050         }
00051
00052         EXPECT_EQ(0, int_test.size());
00053         EXPECT_EQ(0, double_test.size());
00054
```

```
00055         for (size_t i = 0; i < 5; ++i) {
00056             int_test.push_back(i);
00057         }
00058         for (size_t i = 0; i < 5; i++)
00059         {
00060             EXPECT_EQ(i, int_test[i]);
00061         }
00062         EXPECT_THROW(int_test[-1], std::range_error);
00063         EXPECT_THROW(int_test[5], std::range_error);
00064 
00065     } END
00066 #endif
00067 
00068 #if TST > 1
00069 
00070     TEST(Date, tesztek) {
00071 
00072         Date a(2023, 05, 14);
00073         EXPECT_TRUE(std::string("2023-05-14")== a.toString());
00074         EXPECT_THROW(Date b(1700, 05, 14), std::range_error);
00075         EXPECT_THROW(Date c(2000, 0, 14), std::range_error);
00076         Date b(2000, 1, 14);
00077         Date c(2000, 1, 14);
00078         EXPECT_TRUE(b == c);
00079         Date d(2000, 2, 14);
00080         Date e(2000, 3, 14);
00081         EXPECT_FALSE(d == e);
00082         EXPECT_TRUE(d>b);
00083         EXPECT_TRUE(b <= d);
00084         EXPECT_TRUE(d>=b);
00085         e.print_date();
00086         std::cout«std::endl;
00087 
00088     }END
00089 
00090 #endif
00091 
00092 #if TST > 2
00093 
00094     TEST(Contract_date, tesztek) {
00095 
00096         Date a(2000, 3, 14);
00097         Contract_date c(2000, 3, 14, 2222, 03, 12);
00098         EXPECT_TRUE(a== c.getBegin());
00099         Date b(2222, 03, 12);
00100         Contract_date d(2000, 3, 14, 2222, 03, 12);
00101         EXPECT_TRUE(b== d.getEnd());
00102         c.setBegin(2222, 03, 12);
00103         EXPECT_TRUE(b == c.getBegin());
00104         d.setEnd(2000, 3, 14);
00105         EXPECT_TRUE(a == d.getEnd());
00106         c.setBegin(a);
00107         EXPECT_TRUE(a == c.getBegin());
00108         Date i(2300, 03, 02);
00109         c.setEnd(i);
00110         EXPECT_TRUE(i == c.getEnd());
00111         EXPECT_THROW(Contract_date e(2400, 3, 14, 2222, 03, 12), std::range_error);
00112         Contract_date f(2000, 3, 14, 2222, 03, 12);
00113         Date g(2000, 3, 15);
00114         EXPECT_TRUE(f.contains(g));
00115         Contract_date h(2020, 3, 14, 2202, 03, 12);
00116         f=h;
00117         EXPECT_TRUE(Date(2020, 3, 14)== f.getBegin());
00118         EXPECT_TRUE(Date(2202, 03, 12)== f.getEnd());
00119         EXPECT_THROW(f.setBegin(2222, 03, 12), std::range_error);
00120         EXPECT_THROW(f.setEnd(2000, 03, 12), std::range_error);
00121         EXPECT_THROW(f.setBegin(Date(2222, 03, 12)), std::range_error);
00122         EXPECT_THROW(f.setEnd(Date(2000, 03, 12)), std::range_error);
00123 
00124     }END
00125 
00126 #endif
00127 
00128 #if TST > 3
00129 
00130     TEST(Client, tesztek) {
00131 
00132         Client* a= new Client("En");
00133         EXPECT_EQ(0, a->getSumOfAll_Contract());
00134         EXPECT_TRUE(std::string("En")== a->getName());
00135         Contract b(a, ContractType::Regular, 2000, 03, 11, 2004, 03, 12, 2003, 12, 22,
      a->getSumOfAll_Contract() + 1, 74);
00136         a->addContract(b);
00137         EXPECT_EQ(1, a->getSumOfAll_Contract());
00138         EXPECT_EQ(1, a->getContract(1).getContractID());
00139         EXPECT_TRUE(std::string("Client name: En")== a->getsub_Client_info());
00140         EXPECT_THROW(a->getContract(0), std::range_error);
```

```
00141         delete a;
00142
00143     }END
00144
00145 #endif
00146
00147 #if TST > 4
00148
00149     TEST(Person, tesztek) {
00150         Person a("En", "167890ke");
00151         EXPECT_STREQ("167890ke", a.getID().c_str());
00152         EXPECT_TRUE("Person name: En, ID: 167890ke"==a.getsub_Client_info());
00153
00154     }END
00155
00156 #endif
00157
00158 #if TST > 5
00159
00160     TEST(Company, tesztek) {
00161
00162         Company a("En", "16789034647");
00163         EXPECT_STREQ("16789034647", a.getID().c_str());
00164         EXPECT_STREQ("Company name: En, Tax ID: 16789034647", a.getsub_Client_info().c_str());
00165
00166     }END
00167
00168 #endif
00169
00170 #if TST > 6
00171
00172     TEST(Contract, tesztek) {
00173
00174         Client* a= new Client("En");
00175         Contract b(a, ContractType::Regular, 2000, 03, 11, 2004, 03, 12, 2003, 12, 22,
      a->getSumOfAll_Contract() + 1, 74);
00176         b.setCtype(ContractType::VIP);
00177         EXPECT_TRUE(ContractType::VIP== b.getCtype());
00178         EXPECT_STREQ("VIP", b.Ctype_toString().c_str());
00179         b.setTariff(84);
00180         EXPECT_EQ(84, b.getTariff());
00181         b.addConsumption(55);
00182         EXPECT_EQ(55, b.getConsumption());
00183         b.setBalance(-66);
00184         EXPECT_EQ(-66, b.getBalance());
00185         b.pay(66);
00186         EXPECT_EQ(0, b.getBalance());
00187         EXPECT_TRUE(Date(2003, 12, 22)== b.getLast_invoicing());
00188         b.setLast_invoicing(Date(2000, 03, 11));
00189         EXPECT_TRUE(Date(2000, 03, 11)== b.getLast_invoicing());
00190         EXPECT_TRUE(a->getName() == b.getClient()->getName());
00191         Contract_date c(2000, 3, 14, 2222, 03, 12);
00192         b.setCtime(c);
00193         EXPECT_TRUE(b.getCtime().getBegin() == c.getBegin() && b.getCtime().getEnd() == c.getEnd());
00194         EXPECT_EQ(1, b.getContractID());
00195         delete a;
00196
00197     }END
00198
00199 #endif
00200
00201 #if TST > 7
00202
00203     TEST(MVM, tesztek) {
00204
00205         MVM mvm;
00206         Person* a = new Person("En", "198245uf");
00207         mvm.addClient(a);
00208         EXPECT_TRUE(a == mvm.getClient("En", "198245uf"));
00209         mvm.addContract(a, ContractType::Regular, 2000, 03, 11, 2004, 03, 12, 2003, 12, 22);
00210         EXPECT_EQ(a->getSumOfAll_Contract()  , mvm.getClient("En",
      "198245uf")->getContract(1).getContractID());
00211         EXPECT_STREQ("Company name: Meseorszagi Villamos Muvek ZRT., Tax ID: 10760798244",
      mvm.getCompanyDetails().c_str());
00212     }END
00213
00214 #endif
00215
00216 #if TST > 8
00217
00218
00219     MVM mvm;
00220     TEST(Fajlkezeles, szamlazas) {
00221         Person* a = new Person("En", "1982451");
00222         mvm.addClient(a);
00223         mvm.addContract(a, ContractType::Regular, 2000, 03, 11, 2004, 03, 12, 2003, 12, 22);
00224         a->save_invoice(1, Date(2004, 02, 14));
```

```
00225     }END
00226     TEST(Fajlkezeles, load_from_console) {
00227         for (size_t i = 0; i < 5; i++)
00228         {
00229             mvm.load_from_console();
00230         }
00231         EXPECT_THROW(mvm.load_from_console(), std::invalid_argument);
00232 
00233     } END
00234     TEST(Fajlkezeles, save) {
00235         EXPECT_NO_THROW(mvm.save());
00236     } END
00237     TEST(Fajlkezeles, load_from_file) {
00238         MVM mvm2;
00239         EXPECT_THROW(mvm2.load_from_save(10), std::ios_base::failure);
00240         EXPECT_NO_THROW(mvm2.load_from_save(1));
00241     } END
00242 
00243 
00244 #endif
00245 
00246     return 0;
00247 
00248 }
```

Index