Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informaikai Kar

Telekommunikációs szolgáltató nyilvántartását modellező program

Szerzők:
Szatmáry Zoltán

2023. április

Tartalomjegyzék

1. Feladat

1.1. Eredeti feladatkírás

Mobilszolgáltató

Egy mobilszolgáltatónál egy egyedi nyilvántartó programmal szeretnék kezelni az ügyfeleket. Az ügyfeleknek van neve és címe, valamint telefonszáma, ami egyben az egyedi azonosítjuk is. A szolgáltató jelenleg három csomagot biztosít ügyfeleinek: Alap, MobiNet és SMSMax, de később több csomag is lehet. Minden csomaghoz más percdj és SMS díj tartozik, valamint a számítás módszere is eltérő lehet. A MobiNet csomag esetén pl. az is megadható, hogy hány SMS-t küldhet az ügyfél ingyen. A program egy fájlból olvassa be az ügyfelek adatait és választott díjcsomagot. Egy másik fájlból pedig az adott hónapban küldött SMS darabszámot és a lebeszélt perceket. A program írja ki, hogy az egyes ügyfelek mennyit fizetnek a forgalom alapján.

1.2. Módosítások

- A beszédperceken és az SMS-eken kívül legyen eltárolva a havi számlánál a mobilinternet-használat is.
- Egy ügyfél számlájának kiállításakor tételesen szerepeljen az összes havi kiadás egyenként is és összegezve is.

1.3. Kiegészítés

Az adatcsomagok az eredeti feladattól eltérően, a következőek szerint legyenek megvalósítva.

1.3.1. Alapcsomagok

Az alapcsomagokhoz tartozzék egy alapdíj, illetve az egyes szolgáltatásokhoz társuljon különböző árú tarifa.

Basic 1990 Ft-os alapdíjért cserébe 30 Ft/perc, illetve 45 Ft/SMS, valamint 500 MB mobilnet, melye a korlát túllépésével 5 Ft/MB tarifává alakul.

ZoomerNet 5990 Ft-os alapdíjért cserébe korlátlan mobilinternet és 15 Ft/perc, illetve sávosított SMS tarifa, mely az első 5 darab SMS-t 45, az afeletti 10 darabot 25, majd minden további SMS-t 5 Ft/SMS tarifával számol.

AllInMax 9990 Ft-os alapdíjért cserébe korlátlan mobilinternet, hívás és SMS.

2. Specifikáció

2.1. Feladatspecifikáció

A program célja, hogy egy telekommunikációs szolgáltató nyilvántartását modellezze, vagyis képes betölteni és kezelni az ügyfelek személyes adatait, az ügyfelek által igényelt adatcsomagokat, illetve az egyes hónapokban összesített adathasználati mérőszámokat az egyes ügyfelekre lebontva.

2.2. Használat

A program futásakor a parancssorról várja az input, illetve output fájlok neveit, melyek sorrendben a következőek.

1. Az ügyfelek rekordjait tároló fájl neve.
2. Az ügyfelek adathasználati rekordjait tároló fájl neve.
3. A készítendő jelentésfájl neve.

2.2.1. Példa használat

```
telco users.txt datausages.txt report.txt
```

2.3. Hibakezelés

A program mindennemű hibát fatálisnak vesz, így a hiba konzolra való kiírása után leállítja a futást, és megszakítja az esetleges fájlba írásokat.

2.4. Adatok formátuma

Mindegyik fájltípus rekordok sorozatából áll, és az egyes rekordok sorai egy-egy üres sorral vannak elválasztva.

Egy ügyfél leírásának első három sora rendre a nevét, a lakcímét és a telefonszámát (ügyfélazonosítóját) tartalmazza, míg a negyedik sora az adatacsomag-összeállítását tartalmazza, vagyis a csomag összetevőinek neveit szóközökkel elválasztva.

Egy adathasználati rekord leírásának első három sora rendre az ügyfél telefonszámát (ügyfélazonosítóját), az adatok rögzítésének hónapját, illetve a haszhnálat mérőszámait (beszédpercek, elküldött SMS-ek száma, mobilinternet-használat) tartalmazza szóközökkel elválasztva.

Egy jelentési rekord első három sora megegyezik az ügyfél leírásának első három sorával, a negyedik sorától kezdődően pedig az ügyfélhez tartozó havi számlarekordok listája következik.

Egy számlarekord első sorában a számlázási hónapot, az azt követő három sorában pedig rendre a beszédpercek, az elküldött SMS-ek, illetve a mobilinternet-használat után fizetendő díjakat tartalmazza forintban, a negyedik sor pedig az összesített fizetendő összeget tartalmazza forintban.

A számlarekordok formázása a következőek szerint történik:

- Minden számlarekord minden sora kétszintű listaformázást alkalmaz.
- A hónapot tartalmazó sor első, a többi második szintű listaelem.
- Az első szinten lévő listaelemek "␣-␣", a második szinten lévő listaelemek pedig "␣␣-␣" prefixet kapnak.

2.4.1. Példa be- és kimenet

```
Tmites Aladár
1117 Budapest, Magyar Tudósok Körútja 2, I épület IE315
+36302401782
AllInMax

Autos Béla
1117 Budapest, Magyar Tudósok Körútja 2, Q épület QB207
+36704206969
ZoomerNet
```

1. ábra. users.txt

```
+36302401782
2023.01.
32 3 4320.1

+36704206969
2022.12.
89 60 128.37

+36302401782
2023.02.
50 6 1500.8
```

2. ábra. datausages.txt

```
Tmites Aladár
1117 Budapest, Magyar Tudósok Körútja 2, I épület IE315
+36302401782
 - 2023.01.
   - fee after minutes: 0 Ft
   - fee after SMS: 0 Ft
   - fee after mobile data: 0 Ft
   - total: 9990 Ft
 - 2023.02.
   - fee after minutes: 0 Ft
   - fee after SMS: 0 Ft
   - fee after mobile data: 0 Ft
   - total: 9990 Ft

Autos Béla
1117 Budapest, Magyar Tudósok Körútja 2, Q épület QB207
+36704206969
...
```

3. ábra. report.txt

3. Terv

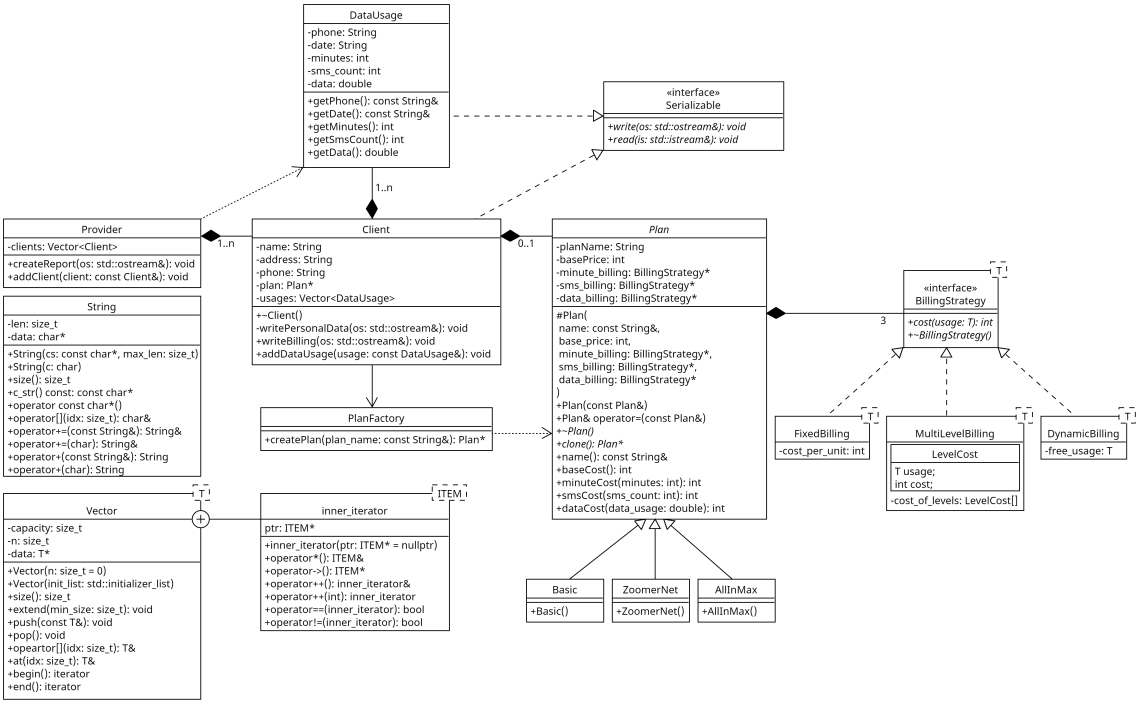
3.1. Objektum terv

3.1.1. Általános terv

Az OOP dekompozíció során tehát mindent elsődlegesen a szolgáltató szempontjából kezdtem el modellezni, vagyis az ilyen szempontból irreleváns részleteket absztraháltam.

Fontosnak láttam, hogy a kód későbbi bővíthetőségét lehetővé tegyem, így az osztályhierachia megvalósításánál olyan már meglévő (létrehozási és viselkedési) tervezési mintákra fogok építeni, mint a *Strategy Pattern* és a *Factory Method Pattern*, melyekkel lehetővé válik az adatcsomagok dinamikus felépítése szöveges leírásból, valamint a felépített csomagok alapján történő számálázás.

A tesztelhetőség érdekében az ügyfelet (*Client*) és a szolgáltatót (*Provider*) reprezentáló osztályokhoz is írok a saját tárolóikba új elemet elhelyező függvényeket, annak ellenére, hogy a főprogram csak a beolvasott adatokon dolgozik, új bejegyzéseket sem az ügyfeleknél, sem az adathasználatoknál nem hoz létre.



4. ábra. Osztálydiagram UML-ben

3.1.2. Az osztályok leírásai

- **Vector**: Egy általánoscélú, sablonos vektor implementáció, mely többek között támogatja a push, a pop és az indexelés műveletét is, valamint rendelkezik konstans és nemkonstans iterátorokkal (**Vector::inner_iterator**) is, így tehát használhatóak rajta az STL algoritmusai és a range-based for loop szintaxisa is.

- **String**: Sztring osztály, mely az STL-ben található std::string-hez nagyon hasonló működést biztosít. Elérhető rajta keresztül a természetes elvárásoknak megfelelően a legtöbb sztringeken értelmezett művelet, mint a karaktertömbből és karakterből való létrehozás, másolás, értékadás, a különböző hozzáfűzések (+, +=), a hasonlító operátorok (==, !=, <, >, <=, >=), illetve free functionként olyan műveletek is, mint a getline vagy az inserter/extractor operátor.
- **DataUsage**: Az ügyfelekhez tartozó havi adathasználati mérőszámokat, illetve a számlázás hónapját és az ügyfelet beazonosító telefonszámot tárolja. Felelőssége csupán az adatok tárolására korlátozódik.
- **Client**: Egy ügyfelél nevét, címét, telefonszámát, választott adatcsomagját, valamint a havi adathasználatait tárolja lista formájában. Felelőssége, hogy kiírassa a személyes adatait és a havi számláit.
- **Plan**: Absztrakt osztály, mely egy tetszőleges adatcsomagot reprezentál. A csomag nevét, egy alapdíját és az egyes adathasználatokhoz rendelt különböző számlázási stratégiákat (*BillingStrategy*) tárolja. Lekérdezhető rajta keresztül az alapdíj, és egyes adathasználati mennyiségek után fizetendő összeg.
- **Basic, ZoomerNet, AllInMax**: A Plan osztály leszármazottjai, tehát a konkrét adatcsomagok implementációi. Felelősségük, hogy beállítsák a a számlázási stratégiákat, illetve a csomag nevét és díjszabását.
- **PlanFactory**: Statikus osztály, mely kizárólagos joggal rendelkezik az adatcsomagok létrehozása felett (ez természetesen az egyedüli felelőssége is).
- **BillingStrategy**: Sablon absztrakt/interfész osztály, mely egy számlázási stratégiát ír le. Egyetlen függvényén keresztül lekérdezhető egy adott adathasználati mennyiség után fizetendő összeg. A sablon megvalósítást azért szükséges, mert a mobil-adathasználat lebegőpontos, a többi mérőszám pedig egész típusú, így nem lehetne ugyanazzal az osztállyal kezelni őket, annak ellenére, hogy az elvégzendő műveletek azonosak, így kódduplikációhoz vezetne, ha külön lennének véve.
- **FixedBilling, MultiLevelBilling, DynamicBilling**: A BillingStrategy osztály leszármazottjai, tehát a specifikáció elvárásainak megfelelő implementációk a számlázási stratégiákra. Ahogy az ősosztály, úgy a leszármazottak is sablonok.
- **MultiLevelBilling::LevelCost**: A többszintű számlázás egy szintjét leíró osztály, mely eltárolja, hogy mekkora adatforgalomig érvényes a szint és milyen tarifa vonatkozik a szintre.

3.2. Algoritmusok

3.2.1. A tesztprogram algoritmusai

A program egészének tesztelését végrehajtó modul legfőbb feladata, hogy a szolgáltató és az ügyfelek adatinak beolvasása után megvizsgálja előre kiszámított adatok alapján, hogy helyesen számolt-e a program.

3.2.2. A jelentéskészítés algoritmusa

```
user ← 1
if user = end then
    return
end if
PrintBilling(user.data)
user ← user + 1
while user ≠ end do
    Print("\n")
    PrintBilling(user.data)
    user ← user + 1
end while
```

4. Tesztelés

4.1. Unit tesztek

Alapvetően osztályokra, valamint azokon belül funkciócsoportokra lebontva lettek megírva a tesztek, így például a Plan osztálynál külön lett választva a példányosítás hatásának vizsgálata és a számlázás helyességének ellenőrzése.

4.2. Stream bemenetek

A tesztek megírásánál fő szempont volt, hogy az adatfolyamként szolgáltatandó bemenetekre is tesztelve legyenek az egyes osztályok, így a Provider osztály tesztelésénél az std::stringstring segítségével van helyettesítve a tényleges adatbemenetről történbő beolvasás.

Telekommunikációs szolgáltató nyilvántartását modellező program

Készítette Doxygen 1.9.5

1. fejezet

Hierarchikus mutató

1.1. Osztályhierarchia

Majdnem (de nem teljesen) betűrendbe szedett leszármazási lista:

2. fejezet

Osztálymutató

2.1. Osztálylista

Az összes osztály, struktúra, unió és interfész listája rövid leírásokkal:

3. fejezet

Osztályok dokumentációja

3.1. AllInMax osztályreferencia

`#include <AllInMax.h>`

Publikus tagfüggvények

- AllInMax ()
- Plan ∗ clone () const override

További örökölt tagok

3.1.1. Részletes leírás

Az AllInMax adatcsomagot megvalósító osztály.

3.1.2. Konstruktorok és destruktorok dokumentációja

3.1.2.1. AllInMax()

`AllInMax::AllInMax ( )`

Beállítja az ősosztály számlázási startégiáit.

3.1.3. Tagfüggvények dokumentációja

3.1.3.1. clone()

```
Plan * AllInMax::clone ( ) const  [override], [virtual]
```

A konkrét adatcsomagot lemásoló függvény.

Visszatérési érték

visszatér az adatcsomag egy dinamikusan foglalt másolatával

Megvalósítja a következőket: Plan.

3.2. Basic osztályreferencia

```
#include <Basic.h>
```

Publikus tagfüggvények

- Basic ()
- Plan * clone () const override

További örökölt tagok

3.2.1. Részletes leírás

A Basic adatcsomagot megvalósító osztály.

3.2.2. Konstruktorok és destruktorok dokumentációja

3.2.2.1. Basic()

```
Basic::Basic ( )
```

Beállítja az ősosztály számlázási startégiáit.

3.2.3. Tagfüggvények dokumentációja

3.2.3.1. clone()

```
Plan * Basic::clone ( ) const  [override], [virtual]
```

A konkrét adatcsomagot lemásoló függvény.

Visszatérési érték

visszatér az adatcsomag egy dinamikusan foglalt másolatával

Megvalósítja a következőket: Plan.

3.3. BillingStrategy< T > osztálysablon-referencia

```
#include <BillingStrategy.hpp>
```

Publikus tagfüggvények

- virtual int cost (T usage) const =0
- virtual ~BillingStrategy ()=default
- virtual BillingStrategy< T > * clone () const =0

3.3.1. Részletes leírás

template<typename T>
class BillingStrategy< T >

Egy számlázási stratégiát leíró osztály.

Sablon paraméterek

T	milyen típusú adathasználat utáni számlázást ír le (pl. int, double)

3.3.2. Konstruktorok és destruktorok dokumentációja

3.3.2.1. ~BillingStrategy()

```
template<typename T >
virtual BillingStrategy< T >::~BillingStrategy ( )  [virtual], [default]
```

Virtuális destruktor, hogy a leszármazottak (erőforrásai) kezelhetőek maradjanak az ősosztály pointerén keresztül is.

3.3.3. Tagfüggvények dokumentációja

3.3.3.1. clone()

```
template<typename T >
virtual BillingStrategy< T > * BillingStrategy< T >::clone ( ) const  [pure virtual]
```

A konkrét számlázási stratégiát lemásoló függvény.

Visszatérési érték

visszatér a számlázási stratégia egy dinamikusan foglalt másolatával

Megvalósítják a következők: DynamicBilling< T >, FixedBilling< T > és MultiLevelBilling< T, LEVEL_COUNT >.

3.3.3.2. cost()

```
template<typename T >
virtual int BillingStrategy< T >::cost (
            T usage ) const  [pure virtual]
```

Kiszámolja, mennyit kell adott adathasználat után fizetni (forintban).

Paraméterek

usage	adathasználat

Visszatérési érték

fizetendő összeg (forintban)

Megvalósítják a következők: DynamicBilling< T >, FixedBilling< T > és MultiLevelBilling< T, LEVEL_COUNT >.

3.4. Client osztályreferencia

```
#include <Client.h>
```

Publikus tagfüggvények

- Client ()
- Client (const utils::String &name, const utils::String &address, const utils::String &phone, const utils::String &plan_name, const utils::Vector< DataUsage > &usages)

- Client (const Client &client)
- Client & operator= (const Client &client)
- ~Client ()
- const utils::String & getPhone () const
- void writePersonalData (std::ostream &os) const
- void writeBilling (std::ostream &os) const
- void addDataUsage (const DataUsage &usage)
- void write (std::ostream &os) const override
- void read (std::istream &is) override

3.4.1. Részletes leírás

Egy ügyfelet leíró osztály. Magában foglalja az ügyfél összes személyes adatát és a havi adathasználatait.

3.4.2. Konstruktorok és destruktorok dokumentációja

3.4.2.1. Client() [1/3]

```
Client::Client ( )
```

Alapértelmezett konstruktor. Az osztály típusú adattagoknak a beépített konstruktorát implicit meghívja, majd az adatcsomagra mutató pointert kezdetben nullptr értékre inicializálja.

3.4.2.2. Client() [2/3]

```
Client::Client (
            const utils::String & name,
            const utils::String & address,
            const utils::String & phone,
            const utils::String & plan_name,
            const utils::Vector< DataUsage > & usages )
```

Értékekkel inicializáló konstruktor.

Paraméterek

name	név
address	cím
phone	telefonszám
plan_name	díjcsomag neve
usages	havi adathasználatok listája

3.4.2.3. Client() [3/3]

```
Client::Client (
            const Client & client )
```

Másoló konstruktor.

Paraméterek

client	másolandó ügyfél

3.4.2.4. ~Client()

```
Client::~Client ( )
```

Destruktor, melynek feladata az adatcsomag felszabadítása.

3.4.3. Tagfüggvények dokumentációja

3.4.3.1. addDataUsage()

```
void Client::addDataUsage (
            const DataUsage & usage )
```

Hozzáad egy adathasználatot a listához.

Kivételt dob, ha az adathasználat nem az ügyféllhez tartozik (vagyis nem egyező telefonszám esetén)!

Paraméterek

usage	adathasználat

Kivételek

std::invalid_argument	

3.4.3.2. getPhone()

```
const utils::String & Client::getPhone ( ) const
```

Visszatér a telefonszámmal.

Visszatérési érték

telefonszám

3.4.3.3. operator=()

```
Client & Client::operator= (
            const Client & client )
```

Értékadó operátor

Paraméterek

client	másolandó ügyfél

Visszatérési érték

a példány referenciája

3.4.3.4. read()

```
void Client::read (
            std::istream & is )  [override], [virtual]
```

Beolvassa a megadott folyamról az objektumot.

Paraméterek

is	bemeneti adatfolyam

Megvalósítja a következőket: Serializable.

3.4.3.5. write()

```
void Client::write (
            std::ostream & os ) const  [override], [virtual]
```

Kiírja a megadott folyamra az objektumot.

Paraméterek

os	kimeneti adatfolyam

Megvalósítja a következőket: Serializable.

3.4.3.6. writeBilling()

```
void Client::writeBilling (
            std::ostream & os ) const
```

Kiírja a kimeneti adatfolyamra az ügyfél számláit.

Paraméterek

os	kimeneti adatfolyam

3.4.3.7. writePersonalData()

```
void Client::writePersonalData (
            std::ostream & os ) const
```

Kiírja a kimeneti adatfolyamra az ügyfél adatait.

Paraméterek

os	kimeneti adatfolyam

3.5. DataUsage osztályreferencia

```
#include <DataUsage.h>
```

Publikus tagfüggvények

- DataUsage (const utils::String &phone="", const utils::String &date="", int minutes=0, int sms_count=0, double data=0.0)
- utils::String const & getPhone () const
- const utils::String & getDate () const
- int getMinutes () const
- int getSmsCount () const
- double getData () const
- void write (std::ostream &os) const override
- void read (std::istream &is) override

3.5.1. Részletes leírás

Egy ügyfél adathasználatát leíró osztály.

3.5.2. Konstruktorok és destruktorok dokumentációja

3.5.2.1. DataUsage()

```
DataUsage::DataUsage (
            const utils::String & phone = "",
            const utils::String & date = "",
            int minutes = 0,
            int sms_count = 0,
            double data = 0.0 )  [explicit]
```

Az adathasználat részleteit beállító konstruktor.

Paraméterek

phone	az ügyfél telefonszáma, akihez az adatok rögzítve lettek
date	a számlázás hónapja
minutes	híváspercek
sms_count	SMS-ek száma
data	belföldi adathasználat (MB-ban)

3.5.3. Tagfüggvények dokumentációja

3.5.3.1. getData()

```
double DataUsage::getData ( ) const
```

Visszatér az adathasználattal.

Visszatérési érték

adathasználat (MB-ban)

3.5.3.2. getDate()

```
const utils::String & DataUsage::getDate ( ) const
```

Visszatér a számlázás hónapjával.

Visszatérési érték

számlázás hónapja

3.5.3.3. getMinutes()

```
int DataUsage::getMinutes ( ) const
```

Visszatér a híváspercekkel.

Visszatérési érték

híváspercek

3.5.3.4. getPhone()

```
utils::String const & DataUsage::getPhone ( ) const
```

Visszatér az ügyfél telefonszámával.

Visszatérési érték

telefonszám

3.5.3.5. getSmsCount()

```
int DataUsage::getSmsCount ( ) const
```

Visszatér az SMS-ek számával.

Visszatérési érték

SMS-ek száma

3.5.3.6. read()

```
void DataUsage::read (
            std::istream & is )  [override], [virtual]
```

Beolvassa a megadott folyamról az objektumot.

Paraméterek

is	bemeneti adatfolyam

Megvalósítja a következőket: Serializable.

3.5.3.7. write()

```
void DataUsage::write (
            std::ostream & os ) const  [override], [virtual]
```

Kiírja a megadott folyamra az objektumot.

Paraméterek

os	kimeneti adatfolyam

Megvalósítja a következőket: Serializable.

3.6. DynamicBilling< T > osztálysablon-referencia

```
#include <DynamicBilling.hpp>
```

Publikus tagfüggvények

- DynamicBilling (T free_usage, int cost_per_unit)
- int cost (T usage) const override
- BillingStrategy< T > * clone () const override

3.6.1. Részletes leírás

template<typename T>
class DynamicBilling< T >

A dinamikus számlázást megvalósító osztály, mely adott egységnyi adathasználatot ingyen lehetővé tesz, és csak az afölötti adathasználat után számláz arányosan.

Sablon paraméterek

T	milyen típusú adathasználat utáni számlázást ír le (pl. int, double)

3.6.2. Konstruktorok és destruktorok dokumentációja

3.6.2.1. DynamicBilling()

```
template<typename T >
DynamicBilling< T >::DynamicBilling (
```

```
            T free_usage,
            int cost_per_unit )  [inline]
```

A konstruktor beállítja a számlázás paramétereit.

Paraméterek

free_usage	ingyenes kvóta
cost_per_unit	tarifa

3.6.3. Tagfüggvények dokumentációja

3.6.3.1. clone()

```
template<typename T >
BillingStrategy< T > * DynamicBilling< T >::clone ( ) const  [inline], [override], [virtual]
```

A konkrét számlázási stratégiát lemásoló függvény.

Visszatérési érték

visszatér a számlázási stratégia egy dinamikusan foglalt másolatával

Megvalósítja a következőket: BillingStrategy< T >.

3.6.3.2. cost()

```
template<typename T >
int DynamicBilling< T >::cost (
            T usage ) const  [inline], [override], [virtual]
```

Kiszámolja, mennyit kell adott adathasználat után fizetni (forintban).

Paraméterek

usage	adathasználat

Visszatérési érték

fizetendő összeg (forintban)

Megvalósítja a következőket: BillingStrategy< T >.

3.7. FixedBilling< T > osztálysablon-referencia

`#include <FixedBilling.hpp>`

Publikus tagfüggvények

- FixedBilling (int cost_per_unit)
- int cost (T usage) const override
- BillingStrategy< T > * clone () const override

3.7.1. Részletes leírás

template<typename T>
class FixedBilling< T >

A fix számlázást megvalósító osztály, mely minden adathasználatot arányosan számláz a tarifa szerint.

Sablon paraméterek

T	milyen típusú adathasználat utáni számlázást ír le (pl. int, double)

3.7.2. Konstruktorok és destruktorok dokumentációja

3.7.2.1. FixedBilling()

```
template<typename T >
FixedBilling< T >::FixedBilling (
            int cost_per_unit )  [inline], [explicit]
```

A konstruktor beállítja a számlázás paramétereit.

Paraméterek

cost_per_unit	tarifa

3.7.3. Tagfüggvények dokumentációja

3.7.3.1. clone()

```
template<typename T >
BillingStrategy< T > * FixedBilling< T >::clone ( ) const  [inline], [override], [virtual]
```

A konkrét számlázási stratégiát lemásoló függvény.

Visszatérési érték

visszatér a számlázási stratégia egy dinamikusan foglalt másolatával

Megvalósítja a következőket: BillingStrategy< T >.

3.7.3.2. cost()

```
template<typename T >
int FixedBilling< T >::cost (
            T usage ) const  [inline], [override], [virtual]
```

Kiszámolja, mennyit kell adott adathasználat után fizetni (forintban).

Paraméterek

usage	adathasználat

Visszatérési érték

fizetendő összeg (forintban)

Megvalósítja a következőket: BillingStrategy< T >.

3.8. MultiLevelBilling< T, LEVEL_COUNT >::LevelCost struktúrareferencia

`#include <MultiLevelBilling.hpp>`

Publikus tagfüggvények

- LevelCost (T usage=0, int cost=0)

Publikus attribútumok

- T usage
- int cost

3.8.1. Részletes leírás

template<typename T, size_t LEVEL_COUNT>
struct MultiLevelBilling< T, LEVEL_COUNT >::LevelCost

Egy szint számlázási paramétereit leíró osztály.

3.8.2. Konstruktorok és destruktorok dokumentációja

3.8.2.1. LevelCost()

```
template<typename T , size_t LEVEL_COUNT>
MultiLevelBilling< T, LEVEL_COUNT >::LevelCost::LevelCost (
            T usage = 0,
            int cost = 0 )  [inline]
```

Beállítja a szint számlázási paramétereit

Paraméterek

usage	mekkora adathasználatig érvényes a szint tarifája
cost	tarifa

3.8.3. Adattagok dokumentációja

3.8.3.1. cost

```
template<typename T , size_t LEVEL_COUNT>
int MultiLevelBilling< T, LEVEL_COUNT >::LevelCost::cost
```

tarifa

3.8.3.2. usage

```
template<typename T , size_t LEVEL_COUNT>
T MultiLevelBilling< T, LEVEL_COUNT >::LevelCost::usage
```

mekkora adathasználatig érvényes a szint tarifája

3.9. MultiLevelBilling< T, LEVEL_COUNT > osztálysablon-referencia

```
#include <MultiLevelBilling.hpp>
```

Osztályok

- struct LevelCost

Publikus tagfüggvények

- MultiLevelBilling (std::initializer_list< LevelCost > init)
- int cost (T usage) const override
- BillingStrategy< T > * clone () const override

3.9.1. Részletes leírás

template<typename T, size_t LEVEL_COUNT>
class MultiLevelBilling< T, LEVEL_COUNT >

A többszintű számlázást megvalósító osztály, mely szintenként más tarifa szerint számláz.

Sablon paraméterek

T	milyen típusú adathasználat utáni számlázást ír le (pl. int, double)
LEVEL_COUNT	hány szintű a számlázás

3.9.2. Konstruktorok és destruktorok dokumentációja

3.9.2.1. MultiLevelBilling()

```
template<typename T , size_t LEVEL_COUNT>
MultiLevelBilling< T, LEVEL_COUNT >::MultiLevelBilling (
            std::initializer_list< LevelCost > init )  [inline]
```

A konstruktor beállítja a számlázás paramétereit, mely a szintenkénti számlázás listája.

Paraméterek

init	inicializáló lista a szintek számlázási paramétereivel

3.9.3. Tagfüggvények dokumentációja

3.9.3.1. clone()

```
template<typename T , size_t LEVEL_COUNT>
BillingStrategy< T > * MultiLevelBilling< T, LEVEL_COUNT >::clone ( ) const  [inline], [override],
[virtual]
```

A konkrét számlázási stratégiát lemásoló függvény.

Visszatérési érték

visszatér a számlázási stratégia egy dinamikusan foglalt másolatával

Megvalósítja a következőket: BillingStrategy< T >.

3.9.3.2. cost()

```
template<typename T , size_t LEVEL_COUNT>
int MultiLevelBilling< T, LEVEL_COUNT >::cost (
            T usage ) const  [inline], [override], [virtual]
```

Kiszámolja, mennyit kell adott adathasználat után fizetni (forintban).

Paraméterek

usage	adathasználat

Visszatérési érték

fizetendő összeg (forintban)

Megvalósítja a következőket: BillingStrategy< T >.

3.10. Plan osztályreferencia

```
#include <Plan.h>
```

Publikus tagfüggvények

- Plan (const Plan &plan)
- Plan & operator= (const Plan &plan)
- virtual ~Plan ()
- virtual Plan * clone () const =0
- const utils::String & name () const
- int baseCost () const
- int minuteCost (int minutes) const
- int smsCost (int sms_count) const
- int dataCost (double data_usage) const

Védett tagfüggvények

- Plan (const utils::String &plan_name, int base_price, BillingStrategy< int > *minute_billing, BillingStrategy< int > *sms_billing, BillingStrategy< double > *data_billing)

3.10.1. Részletes leírás

A Plan egy absztrakt osztály, ami definiálja a díjcsomagok közös tulajdonságait és interfészét.

3.10.2. Konstruktorok és destruktorok dokumentációja

3.10.2.1. Plan() [1/2]

```
Plan::Plan (
            const utils::String & plan_name,
            int base_price,
            BillingStrategy< int > * minute_billing,
            BillingStrategy< int > * sms_billing,
            BillingStrategy< double > * data_billing )  [protected]
```

Csak a leszármazottak számára elérhető konstruktor, amely beállítja a csomag adatait.

3.10.2.2. Plan() [2/2]

```
Plan::Plan (
            const Plan & plan )
```

Másoló konstruktor.

Paraméterek

plan	másolandó adatcsomag

3.10.2.3. ∼Plan()

```
Plan::∼Plan ( )  [virtual]
```

Virtuális destruktor, hogy a leszármazott objektumokat heterogén kollekcióként lehessen kezelni. Felszabadítja továbbá a számlázási stratégiákat.

3.10.3. Tagfüggvények dokumentációja

3.10.3.1. baseCost()

```
int Plan::baseCost ( ) const
```

Visszatér a csomag alap díjszabásával.

Visszatérési érték

fizetendő összeg (forintban)

3.10.3.2. clone()

```
virtual Plan * Plan::clone ( ) const  [pure virtual]
```

A konkrét adatcsomagot lemásoló függvény.

Visszatérési érték

visszatér az adatcsomag egy dinamikusan foglalt másolatával

Megvalósítják a következők: AllInMax, Basic és ZoomerNet.

3.10.3.3. dataCost()

```
int Plan::dataCost (
            double data_usage ) const
```

Kiszámolja egy ügyfél által az mobil-adathasználat után fizetendő összeget a díjcsomag alapján.

Paraméterek

data_usage	mobil-adathasználat

Visszatérési érték

fizetendő összeg (forintban)

3.10.3.4. minuteCost()

```
int Plan::minuteCost (
            int minutes ) const
```

Kiszámolja egy ügyfél által a hívások után fizetendő összeget a díjcsomag alapján.

Paraméterek

minutes	híváspercek

Visszatérési érték

fizetendő összeg (forintban)

3.10.3.5. name()

```
const utils::String & Plan::name ( ) const
```

Visszaadja a csomag nevét.

Visszatérési érték

csomag neve

3.10.3.6. operator=()

```
Plan & Plan::operator= (
            const Plan & plan )
```

Értékadó operátor

Paraméterek

plan	másolandó adatcsomag

Visszatérési érték

a példány referenciája

3.10.3.7. smsCost()

```
int Plan::smsCost (
            int sms_count ) const
```

Kiszámolja egy ügyfél által elküldött SMS-ek után fizetendő összeget a díjcsomag alapján.

Paraméterek

sms_count	SMS-ek száma

Visszatérési érték

fizetendő összeg (forintban)

3.11. PlanFactory osztályreferencia

`#include <PlanFactory.h>`

Statikus publikus tagfüggvények

- static Plan * createPlan (const utils::String &plan_name)

3.11.1. Részletes leírás

Az adatcsomagok létrehozásáért felelős osztály.

3.11.2. Tagfüggvények dokumentációja

3.11.2.1. createPlan()

```
Plan * PlanFactory::createPlan (
            const utils::String & plan_name )  [static]
```

Létrehozza dinamikusan a megfelelő adatcsomagot `plan_name` alapján.

Ha hibás, nem létező csomagra hivatkozik a név, kivételt dob!

Paraméterek

plan_name	adatcsomag neve

Visszatérési érték

a dinamikusan létrehozott adatcsomag, melynek felszabadítása a hívó felelőssége

Kivételek

std::invalid_argument	

3.12. Provider osztályreferencia

`#include <Provider.h>`

Publikus tagfüggvények

- Provider (std::istream &client_is=std::cin, std::istream &usage_is=std::cin)
- void createReport (std::ostream &os=std::cout)
- void addClient (const Client &client)

3.12.1. Részletes leírás

A szolgáltatót reprezentáló osztály tárolja a díjcsomagokat, valamint az ügyfelek adatait és adatforgalmát.

3.12.2. Konstruktorok és destruktorok dokumentációja

3.12.2.1. Provider()

```
Provider::Provider (
            std::istream & client_is = std::cin,
            std::istream & usage_is = std::cin )  [explicit]
```

A konstruktor beolvassa a `client_is` és a `usage_is` adatfolyamokról a szolgáltató ügyfeleit, valamint a hozzájuk tartozó adathasználatokat.

Paraméterek

client_is	ügyfeleket megadó bemeneti adatfolyam
usage_is	adathasználatokat megadó bemeneti adatfolyam

3.12.3. Tagfüggvények dokumentációja

3.12.3.1. addClient()

```
void Provider::addClient (
            const Client & client )
```

Hozzáad a szolgáltatóhoz egy új ügyfelet.

Paraméterek

client	ügyfél

3.12.3.2. createReport()

```
void Provider::createReport (
            std::ostream & os = std::cout )
```

Kilistázza a szolgáltató ügyfeleit a hozzájuk tartozó adathasználat után fizetendő összeggel együtt.

Paraméterek

os	kimeneti adatfolyam

3.13. Serializable osztályreferencia

```
#include <Serializable.h>
```

Publikus tagfüggvények

- virtual ~Serializable ()=default
- virtual void write (std::ostream &os) const =0
- virtual void read (std::istream &is)=0

3.13.1. Részletes leírás

Szerializálható (folyamra kírható/folyamról beolvasható) típusok közös interfésze.

3.13.2. Konstruktorok és destruktorok dokumentációja

3.13.2.1. ~Serializable()

```
virtual Serializable::~Serializable ( )  [virtual], [default]
```

Virtuális destruktor, hogy a leszármazott objektumokat heterogén kollekcióként lehessen kezelni.

3.13.3. Tagfüggvények dokumentációja

3.13.3.1. read()

```
virtual void Serializable::read (
            std::istream & is )  [pure virtual]
```

Beolvassa a megadott folyamról az objektumot.

Paraméterek

is	bemeneti adatfolyam

Megvalósítják a következők: Client és DataUsage.

3.13.3.2. write()

```
virtual void Serializable::write (
            std::ostream & os ) const  [pure virtual]
```

Kiírja a megadott folyamra az objektumot.

Paraméterek

os	kimeneti adatfolyam

Megvalósítják a következők: Client és DataUsage.

3.14. utils::String osztályreferencia

```
#include <String.h>
```

Publikus tagfüggvények

- String (const char *cs="", size_t max_len=SIZE_MAX)
- String (char c)
- String (const String &rhs)
- ~String ()
- String & operator= (String s)
- size_t size () const
- const char * c_str () const
- operator const char * () const

- char & operator[ ] (size_t idx)
- const char & operator[ ] (size_t idx) const
- String & operator+= (const String &s)
- String & operator+= (char c)
- String operator+ (const String &s) const
- String operator+ (char c) const
- bool operator== (const char ∗cs) const
- bool operator!= (const char ∗cs) const
- bool operator< (const char ∗cs) const
- bool operator<= (const char ∗cs) const
- bool operator> (const char ∗cs) const
- bool operator>= (const char ∗cs) const

3.14.1. Részletes leírás

Egy dinamikusan növelhető sztringet megvalósító osztály.

3.14.2. Konstruktorok és destruktorok dokumentációja

3.14.2.1. String() [1/3]

```
utils::String::String (
            const char * cs = "",
            size_t max_len = SIZE_MAX )
```

Alapértelmezett és C-sztring konverziós konstruktor.

Paraméterek

cs	C-sztring
max_len	a maximálisan felhasználandó hossza `cs` -nek

3.14.2.2. String() [2/3]

```
utils::String::String (
            char c )
```

Karakter konverziós konstruktor.

Paraméterek

c	karakter

3.14.2.3. String() [3/3]

```
utils::String::String (
            const String & rhs )
```

Másoló konstruktor.

Paraméterek

rhs	a másolandó példány

3.14.2.4. ∼String()

```
utils::String::~String ( )
```

Destruktor, melynek felelőssége, hogy felszabadítsa a dinamikusan létrehozott `data` -t.

3.14.3. Tagfüggvények dokumentációja

3.14.3.1. c_str()

```
const char * utils::String::c_str ( ) const
```

Visszatér a sztring belső reprezentációjával, mint C-sztringgel.

Visszatérési érték

C-sztring

3.14.3.2. operator const char ∗()

```
utils::String::operator const char * ( ) const
```

C-sztringgé kasztoló operátor

Visszatérési érték

C-sztring

3.14.3.3. operator"!=()

```
bool utils::String::operator!= (
            const char * cs ) const
```

Egyenlőtlenség-vizsgálat sztringgel.

Paraméterek

cs	C-sztring

Visszatérési érték

nem egyenlő-e `cs-sel`

3.14.3.4. operator+() [1/2]

```
String utils::String::operator+ (
            char c ) const
```

Új sztringet készítő konkatenáló operátor (sztring + karakter).

Paraméterek

c	a példányhoz hozzáfűzendő karakter

Visszatérési érték

új sztring

3.14.3.5. operator+() [2/2]

```
String utils::String::operator+ (
            const String & s ) const
```

Új sztringet készítő konkatenáló operátor (sztring + sztring).

Paraméterek

s	a példányhoz hozzáfűzendő sztring referenciája

Visszatérési érték

új sztring

3.14.3.6. operator+=() [1/2]

```
String & utils::String::operator+= (
            char c )
```

Láncolható konkatenáló operátor (sztring + karakter).

Paraméterek

c	a példányhoz hozzáfűzendő karakter

Visszatérési érték

a példány referenciája

3.14.3.7. operator+=() [2/2]

```
String & utils::String::operator+= (
            const String & s )
```

Láncolható konkatenáló operátor (sztring + sztring).

Paraméterek

s	a példányhoz hozzáfűzendő sztring referenciája

Visszatérési érték

a példány referenciája

3.14.3.8. operator<()

```
bool utils::String::operator< (
            const char * cs ) const
```

Kisebbség-vizsgálat sztringgel.

Paraméterek

cs	C-sztring

Visszatérési érték

kisebb-e `cs`-nél

3.14.3.9. operator<=()

```
bool utils::String::operator<= (
            const char * cs ) const
```

"Kisebbség vagy egyenlőség"-vizsgálat sztringgel.

Paraméterek

cs	C-sztring

Visszatérési érték

kisebb vagy egyenlő-e `cs-sel`

3.14.3.10. operator=()

```
String & utils::String::operator= (
            String s )
```

Az érékadás operátor megvalósítása copy-and-swap módszerrel.

Paraméterek

s	az érték szerint átvett sztring, amellyel egyenlővé tenni való a példány

Visszatérési érték

referencia a példányra

3.14.3.11. operator==()

```
bool utils::String::operator== (
            const char * cs ) const
```

Egyenlőség-vizsgálat sztringgel.

Paraméterek

cs	C-sztring

Visszatérési érték

egyenlő-e `cs-sel`

3.14.3.12. operator>()

```
bool utils::String::operator> (
            const char * cs ) const
```

Nagyobbság-vizsgálat sztringgel.

Paraméterek

cs	C-sztring

Visszatérési érték

nagyobb-e `cs`-nél

3.14.3.13. operator>=()

```
bool utils::String::operator>= (
            const char * cs ) const
```

"Nagyobb vagy egyenlőség"-vizsgálat sztringgel.

Paraméterek

cs	C-sztring

Visszatérési érték

nagyobb vagy egyenlő-e `cs`-sel

3.14.3.14. operator[]() [1/2]

```
char & utils::String::operator[] (
            size_t idx )
```

Indexelő operátor, mely visszatér a sztring `idx` -edik karakterére mutató referenciával.

Nem hajt végre futási idejű ellenőrzést a túlindexelés elkerülésére!

Paraméterek

idx	index

Visszatérési érték

referencia a karakterre

3.14.3.15. operator[]() [2/2]

```
const char & utils::String::operator[] (
            size_t idx ) const
```

Indexelő operátor, mely visszatér a konstans sztring `idx` -edik karakterére mutató konstans referenciával.

Nem hajt végre futási idejű ellenőrzést a túlindexelés elkerülésére!

Paraméterek

idx	index

Visszatérési érték

referencia a karakterre

3.14.3.16. size()

```
size_t utils::String::size ( ) const
```

Visszatér a szöveg hosszával.

Visszatérési érték

szöveg hossza

3.15. utils::Vector< T > osztálysablon-referencia

`#include <Vector.hpp>`

Publikus típusok

- using iterator = inner_iterator< T >
- using const_iterator = inner_iterator< const T >

Publikus tagfüggvények

- Vector (size_t n=0)
- Vector (std::initializer_list< T > init_list)
- Vector (Vector const &rhs)
- Vector & operator= (Vector vector)
- ~Vector ()
- size_t size () const
- void extend (size_t min_size)
- void push (const T &item)
- void pop ()
- T & operator[ ] (size_t idx)
- const T & operator[ ] (size_t idx) const
- T & at (size_t idx)
- const T & at (size_t idx) const
- iterator begin ()
- iterator end ()
- const_iterator begin () const
- const_iterator end () const

Védett attribútumok

- size_t capacity
- size_t n
- T * data

3.15.1. Részletes leírás

template<typename T>
class utils::Vector< T >

Általános célú, vektort (másnéven dinamikus tömböt) megvalósító sablon.

Sablon paraméterek

T	a tárolandó elemek típusa

3.15.2. Típusdefiníció-tagok dokumentációja

3.15.2.1. const_iterator

```
template<typename T >
using utils::Vector< T >::const_iterator = inner_iterator<const T>
```

A konstans példány bejárója, mely a sablonos általános bejáró egy specializációja konstans `T` típusú elemekre.

3.15.2.2. iterator

```
template<typename T >
using utils::Vector< T >::iterator = inner_iterator<T>
```

A nem konstans példány bejárója, mely a sablonos általános bejáró egy specializációja `T` típusú elemekre.

3.15.3. Konstruktorok és destruktorok dokumentációja

3.15.3.1. Vector() [1/3]

```
template<typename T >
utils::Vector< T >::Vector (
            size_t n = 0 )  [inline], [explicit]
```

Az explicit egy paraméteres és egyben alapértelmezett konstruktor egy `n` méretű és egyben kapacitású vektort inicializál.

Paraméterek

n	méret

3.15.3.2. Vector() [2/3]

```
template<typename T >
utils::Vector< T >::Vector (
            std::initializer_list< T > init_list )  [inline]
```

Az inicializáló listás konstruktor, mely az adattagok inicializálását delegálja az egy paraméteres, méret szerint inicializáló konstruktornak, majd átmásolja az adatokat az inicializáló listáról a fizikai tárolóba.

Paraméterek

init_list	inicializáló lista

3.15.3.3. Vector() [3/3]

```
template<typename T >
utils::Vector< T >::Vector (
            Vector< T > const & rhs )  [inline]
```

A másoló konstruktor megvalósítása.

Paraméterek

rhs	a másolandó példányra mutató konstans referencia

3.15.3.4. ~Vector()

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

A destruktor megvalósítása, mely felszabadítja a fizikai tárolót, és azzal együtt az érték szerint tárolt elemeket.

3.15.4. Tagfüggvények dokumentációja

3.15.4.1. at() [1/2]

```
template<typename T >
T & utils::Vector< T >::at (
            size_t idx )  [inline]
```

Visszatér a tároló `idx` -edik elemére mutató referenciával, amennyiben az létezik.

Az index helyességének ellenőrzése (boundary-check) biztosított! Túlindexelés esetén kivételt dob!

Paraméterek

idx	index

Visszatérési érték

referencia az elemre

Kivételek

std::out_of_range	

3.15.4.2. at() [2/2]

```
template<typename T >
const T & utils::Vector< T >::at (
            size_t idx ) const  [inline]
```

Visszatér a konstans tároló `idx` -edik elemére mutató konstans referenciával, amennyiben az létezik.

Az index helyességének ellenőrzése (boundary-check) biztosított! Túlindexelés esetén kivételt dob!

Paraméterek

idx	index

Visszatérési érték

konstans referencia az elemre

Kivételek

std::out_of_range	

3.15.4.3. begin() [1/2]

```
template<typename T >
iterator utils::Vector< T >::begin ( )  [inline]
```

Visszatér a tároló kezdetét jelző bejáróval.

Visszatérési érték

kezdő iterátor

3.15.4.4. begin() [2/2]

```
template<typename T >
const_iterator utils::Vector< T >::begin ( ) const  [inline]
```

Visszatér a konstans tároló kezdetét jelző bejáróval.

Visszatérési érték

kezdő iterátor

3.15.4.5. end() [1/2]

```
template<typename T >
iterator utils::Vector< T >::end ( )  [inline]
```

Visszatér a tároló végét jelző bejáróval.

Visszatérési érték

befejező iterátor

3.15.4.6. end() [2/2]

```
template<typename T >
const_iterator utils::Vector< T >::end ( ) const  [inline]
```

Visszatér a konstans tároló végét jelző bejáróval.

Visszatérési érték

befejező iterátor

3.15.4.7. extend()

```
template<typename T >
void utils::Vector< T >::extend (
            size_t min_size )  [inline]
```

Biztosítja, hogy a vektor mindenképp legyen képes `min_size` sok elem tárolására (az tároló újraallkolásának mellőzésével).

A vektor méretét nem növeli, csak a kapacitását (és azzal együtt a fizikai tárolót is), ha szükséges.

Paraméterek

min_size	az igénylendő mennyiség

3.15.4.8. operator=()

```
template<typename T >
Vector & utils::Vector< T >::operator= (
            Vector< T > vector )  [inline]
```

Az érékadás operátor megvalósítása copy-and-swap módszerrel.

Paraméterek

vector	az érték szerint átvett vektor, amellyel egyenlővé tenni való a példány

Visszatérési érték

referencia a példányra

3.15.4.9. operator[]() [1/2]

```
template<typename T >
T & utils::Vector< T >::operator[] (
            size_t idx )  [inline]
```

Indexelő operátor, mely visszatér a tároló `idx` -edik elemére mutató referenciával.

Nem hajt végre futási idejű ellenőrzést a túlindexelés elkerülésére!

Paraméterek

idx	index

Visszatérési érték

referencia az elemre

3.15.4.10. operator[]() [2/2]

```
template<typename T >
const T & utils::Vector< T >::operator[] (
            size_t idx ) const  [inline]
```

Indexelő operátor a konstans példányra, mely visszatér a tároló `idx` -edik elemére mutató konstans referenciával.

Nem hajt végre futási idejű ellenőrzést a túlindexelés elkerülésére!

Paraméterek

idx	index

Visszatérési érték

konstans referencia az elemre

3.15.4.11. pop()

```
template<typename T >
void utils::Vector< T >::pop ( )  [inline]
```

Kiveszi az utolsó elemet a vektorból.

Nem ellenőrzi, hogy van-e még benne elem!

3.15.4.12. push()

```
template<typename T >
void utils::Vector< T >::push (
            const T & item )  [inline]
```

Betesz egy elemet a vektorba.

Paraméterek

item	az elem

3.15.4.13. size()

```
template<typename T >
size_t utils::Vector< T >::size ( ) const  [inline]
```

Visszaadja a tároló ténylegesen kihasznált méretét.

Visszatérési érték

a méret

3.15.5. Adattagok dokumentációja

3.15.5.1. capacity

```
template<typename T >
size_t utils::Vector< T >::capacity  [protected]
```

a fizikai tároló kapacitása

3.15.5.2. data

```
template<typename T >
T* utils::Vector< T >::data  [protected]
```

a fizikai tároló

3.15.5.3. n

```
template<typename T >
size_t utils::Vector< T >::n  [protected]
```

a fizikai tároló tényleges mérete

3.16. ZoomerNet osztályreferencia

```
#include <ZoomerNet.h>
```

Publikus tagfüggvények

- ZoomerNet ()
- Plan ∗ clone () const override

További örökölt tagok

3.16.1. Részletes leírás

A ZoomerNet adatcsomagot megvalósító osztály.

3.16.2. Konstruktorok és destruktorok dokumentációja

3.16.2.1. ZoomerNet()

```
ZoomerNet::ZoomerNet ( )
```

Beállítja az ősosztály számlázási startégiáit.

3.16.3. Tagfüggvények dokumentációja

3.16.3.1. clone()

```
Plan * ZoomerNet::clone ( ) const  [override], [virtual]
```

A konkrét adatcsomagot lemásoló függvény.

Visszatérési érték

visszatér az adatcsomag egy dinamikusan foglalt másolatával

Megvalósítja a következőket: Plan.

Tárgymutató