

PROGRAMOZÁS

Objektum orientált modellezés

Gregorics Tibor

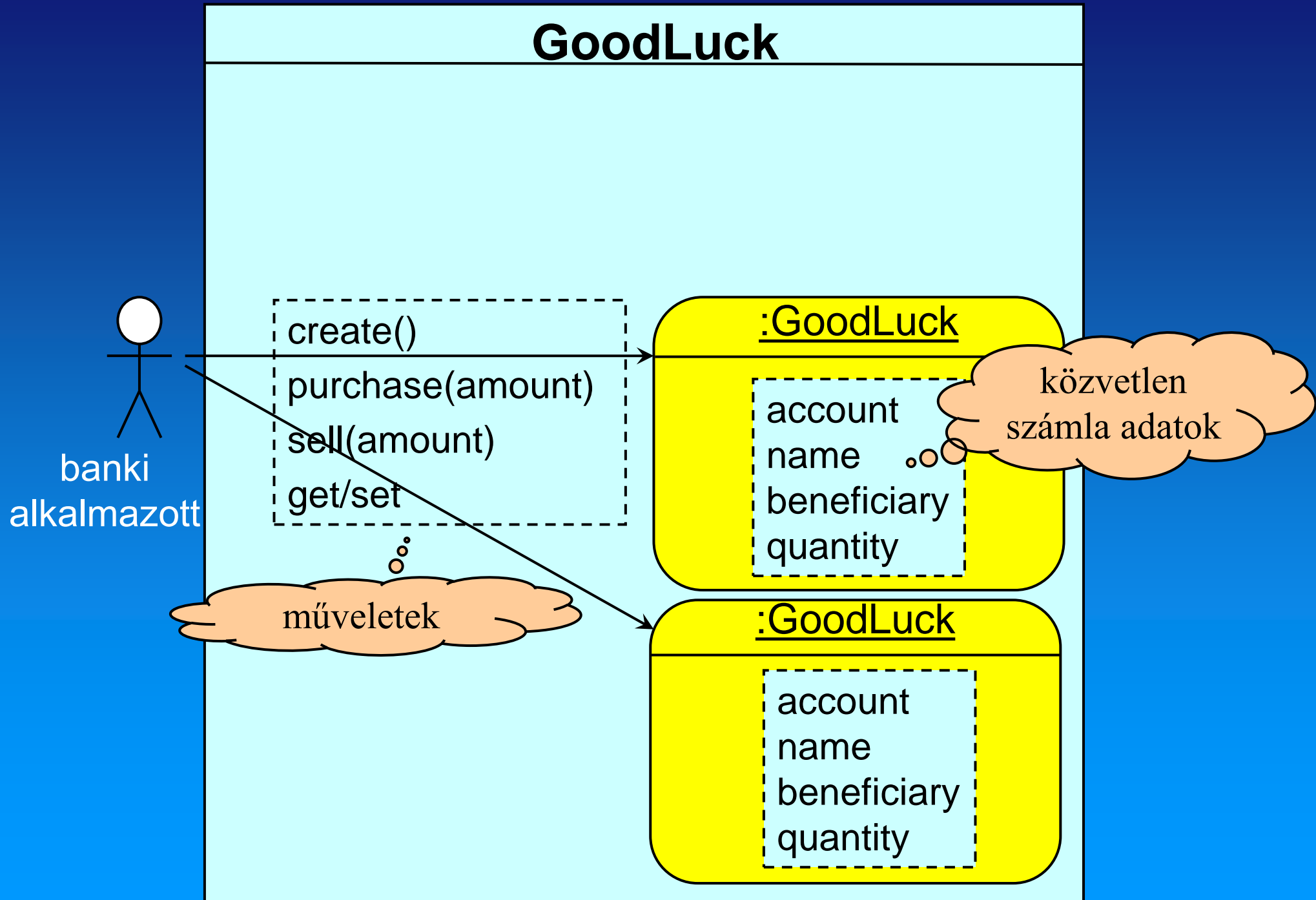
<http://people.inf.elte.hu/gt/prog>

1. Feladat

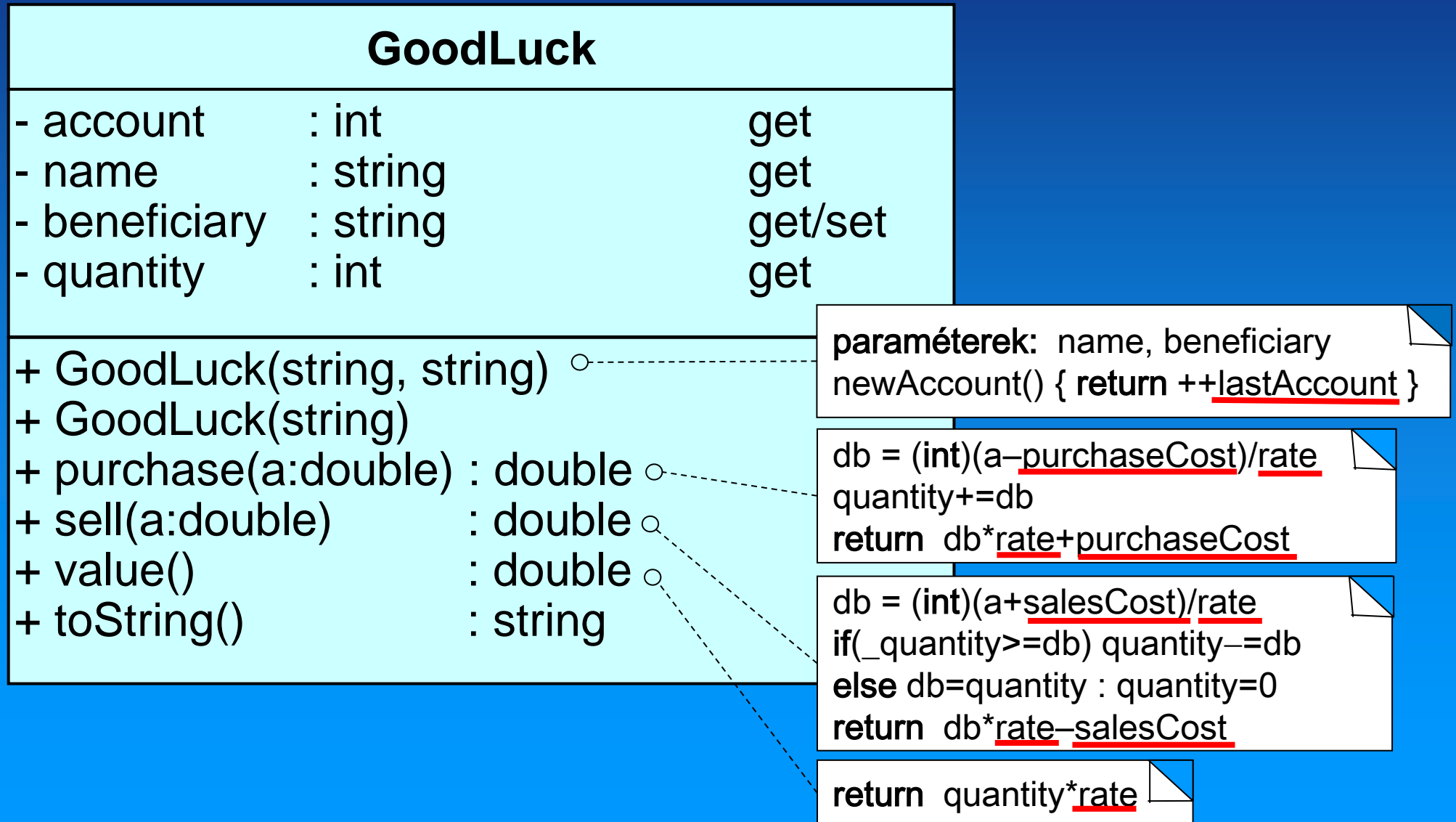
A “GoodLuck” fantázianevű befektetési 5 jegyek forgalmazásánál egy ügyfél számlát 5 nyithat, majd befektetési jegyeket vásárolhat vagy eladhat. A számla nyitásakor egy egyedi számlaszámot kell generálni, meg kell adni a számlatulajdonos nevét, és megadható egy kedvezményezett neve is (azé, aki örökli a számlát). A befektetési jegyek száma a számla nyitásakor nulla. A befektetési jegyek vételi és eladási ára annak árfolyamától, illetve kezelési költségétől függ, amelyek változhatnak.

Készítsünk olyan programot, amely egy-két ügyfél “GoodLuck” befektetési jegyeit forgalmazó számlákat tud kezelni!

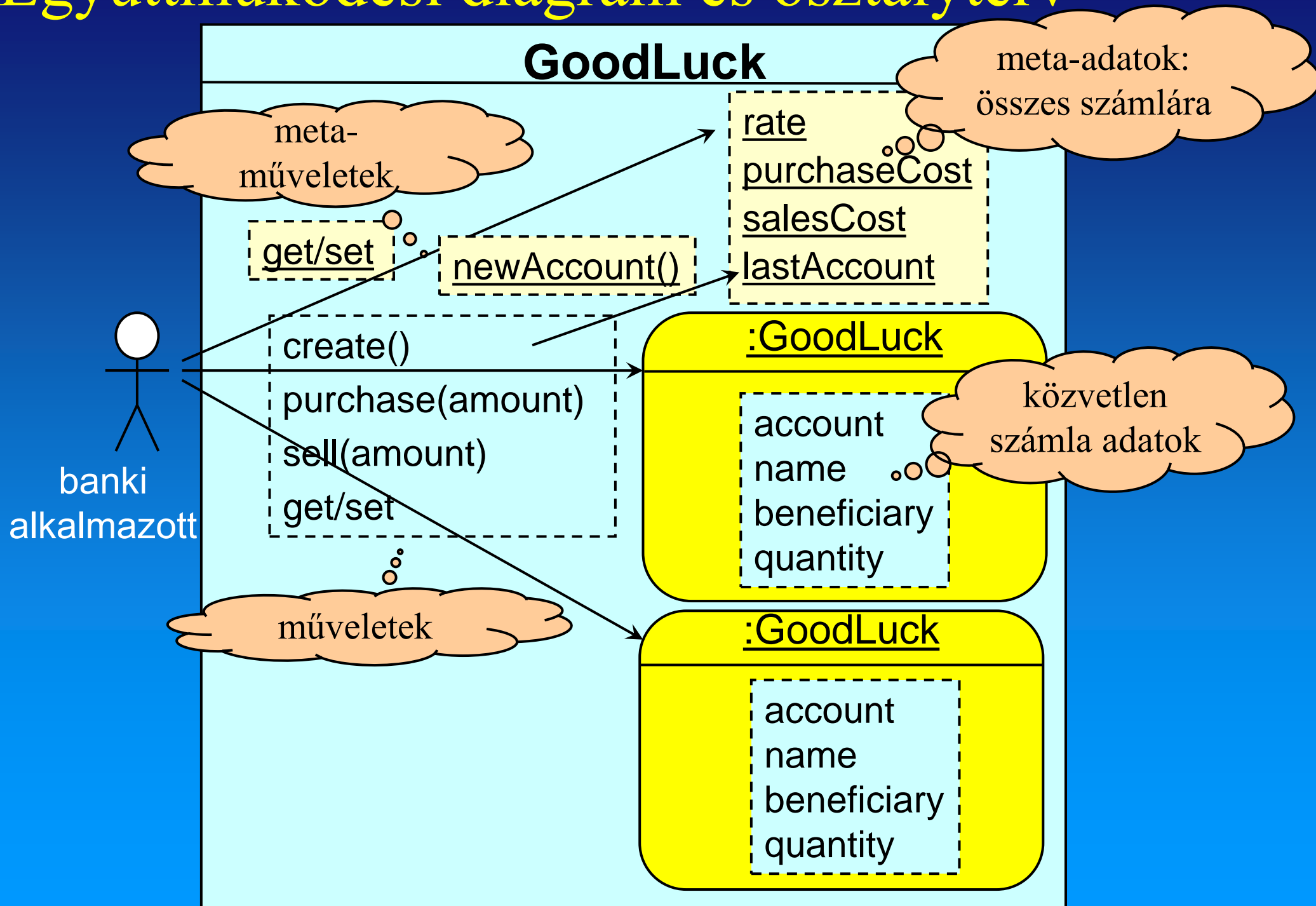
Együttműködési diagram és osztályterv



Osztálydiagram



Együttműködési diagram és osztályterv



Osztálydiagram

GoodLuck

- lastAccount : int = 1000 get
- rate : double = 1.3 get/set
- salesCost : double = 400.0 get/set
- purchaseCost:double = 200.0 get/set

- account : int get
- name : string get
- beneficiary : string get/set
- quantity : int get

+ newAccount() : int
+ howManyUnits(a:double): int

return ++lastAccount

return a/rate

+ GoodLuck(string, string)
+ GoodLuck(string)
+ purchase(a:double) : double
+ sell(a:double) : double
+ value() : double
+ toString() : string

Objektumdiagram

lastAccount newAccount()

g1 : GoodLuck

purchaseCost

: GoodLuck

g2 : GoodLuck

salesCost

rate

: GoodLuck

howManyUnits()

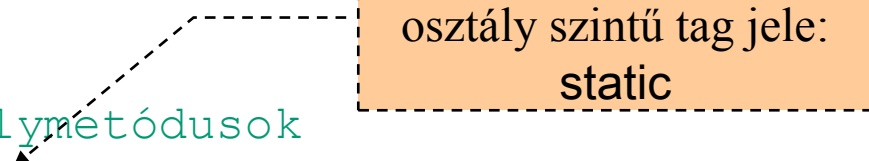
```
class GoodLuck {  
private:  
    // osztály adatainak  
    static int      _lastAccount;  
    static double   _rate;  
    static double   _salesCost;  
    static double   _purchaseCost;  
  
    // példány adatainak  
    int             _account;  
    std::string     _name,  
                    _beneficiary;  
    int             _quantity;  
    ...  
};
```

osztály szintű tag jele:
static


```
...
public:
    // osztálymetódusok
    static int newAccount();
    static int howManyUnits(double amount);

    // konstruktorok
    public GoodLuck(const std::string &str1,
                    const std::string &str2);
    public GoodLuck(const std::string &str);

    // példánymetódusok
    double value() const;
    double purchase(double amount);
    double sell(double amount);
    std::string toString() const;
...
```



osztály szintű tag jele:
static

nem minden adattaghoz
készül set() metódus

```
...  
// hozzáférés az osztály adattagokhoz  
static int      getLastAccount()      { return _lastAccount; }  
static double   getRate()              { return _rate; }  
static void     setRate(double c)      { _rate = c; }  
static double   getSalesCost()         { return _salesCost; }  
static void     setSalesCost(double c) { _salesCost = c; }  
static double   getPurchaseCost()      { return _purchaseCost; }  
static void     setPurchaseCost(double c) { _purchaseCost = c; }  
  
// hozzáférés a példány adattagokhoz  
int getAccount()          const { return _account; }  
std::string getName()     const { return _name; }  
int getQuantity()         const { return _quantity; }  
std::string getBeneficiary() const { return _beneficiary; }  
void setBeneficiary(std::string str)  { _beneficiary = str; }  
};
```

osztály szintű adattagok
kezdeti értékadásai

// osztály adattagok

```
int    GoodLuck::_lastAccount = 1000;
double GoodLuck::_rate = 1.3;
double GoodLuck::_salesCost = 400.0;
double GoodLuck::_purchaseCost = 200.0;
```

// osztálymetódusok

```
int GoodLuck::howManyUnits(double amount)
{
    return (int) ( amount/_rate );
}
```

osztály szintű metódusok nem
hivatkozhatnak közvetlenül
példány szintű elemekre

```
int GoodLuck::newAccount()
{
    return ++_lastAccount;
}
```

...

```
...
// konstruktorok
GoodLuck::GoodLuck(std::string str1, std::string str2)
{
    _account = newAccount();
    _name = str1;
    _beneficiary = str2;
    _quantity = 0;
}

GoodLuck::GoodLuck(std::string str)
{
    _account = newAccount();
    _name = str;
    _beneficiary = "";
    _quantity = 0;
}
...
```

// példánymetódusok

```
double GoodLuck::value() const { return _quantity*_rate; }
```

```
double GoodLuck::purchase(double amount)
{
    int db = howManyUnits(amount-_purchaseCost);
    _quantity += db;
    return db*_rate+_purchaseCost;
}
```

```
double GoodLuck::sell(double amount)
{
    int db = howManyUnits(amount+_salesCost);
    if(_quantity>=db) _quantity -= db;
    else { db = _quantity; _quantity = 0;}
    return db*_rate-_salesCost;
}
```

```
std::string GoodLuck::toString() const
{
    std::ostringstream ss;
    ss << "Számiaszám: " << _account <<
        "Név: " << _name <<
        "Kedvezményezett: " << _beneficiary <<
        "jegyek száma: " << _quantity;
    return ss.str();
}
```

vegyes típusú elemekből
összeszerkeszthető sztring
`#include <sstream>`

goodluck.cpp

Használati forgatókönyv

```
int main()
{
    GoodLuck g1("Bólyai Farkas", "Bólyai János");
    GoodLuck g2("Neumann János");
    g2.setBeneficiary("Neumann Jánosné");

    cout << g1.toString() << endl << g2.toString() << endl;

    cout << "Hány jegyet ér: " << GoodLuck::howManyUnits(5000) << endl;
    cout << "Vásárol: " << g1.purchase(5000) << endl;
    cout << "árfolyam: " << GoodLuck::getRate() << endl;
    cout << "vásárlási költség: " << GoodLuck::getPurchaseCost() << endl;
    GoodLuck::setRate(5.4); GoodLuck::setPurchaseCost(600.0);

    cout << "Vagyon: " << g1.value() << endl;
    cout << "Elad: " << g1.sell(5000) << endl;
    cout << "eladási költség: " << GoodLuck::getSalesCost() << endl;
    GoodLuck::setSalesCost(100.0);

    cout << g1.toString() << endl << g2.toString() << endl;

    return 0;
}
```

osztály szintű
metódus hívása

2. Feladat

A "GoodLuck" és „BadLuck” fantázianevű befektetési jegyek forgalmazásánál egy ügyfél számlát nyithat, majd befektetési jegyeket vásárolhat vagy eladhat. A számla nyitásakor egy egyedi számlaszámot kell generálni, meg kell adni a számlatulajdonos nevét, a "GoodLuck" esetében megadható egy kedvezményezett neve is. A befektetési jegyek száma a számla nyitásakor nulla. A befektetési jegyek vételi és eladási ára annak pillanatnyi árfolyamától, illetve a kezelési költségétől függ, amelyek változtathatóak.

Készítsünk olyan programot, amely egy-két ügyfél befektetési jegyeit forgalmazó számlákat tud kezelni!

Használati forgatókönyv

```
int main()
{
    GoodLuck g("Bólyai Farkas", "Bólyai János");
    BadLuck b("Riesz Frigyes");

    cout << "GoodLuck árfolyam: " << GoodLuck::getRate() << endl;
    cout << "BaddLuck árfolyam: " << BadLuck::getRate() << endl;

    cout << "Vásárol: " << g.purchase(5000) << endl;
    cout << "Vásárol: " << b.purchase(5000) << endl << endl;

    cout << g.toString() << endl << b.toString() << endl;

    GoodLuck::setRate(8.9);
    BadLuck::setRate(1.2);

    cout << "Elad: " << g.sell(5000) << endl;
    cout << "Elad: " << b.sell(2000) << endl;

    cout << g.toString() << endl << b.toString() << endl;

    return 0;
}
```

árfolyam lekérdezés

árfolyam változtatás

1. próbálkozás

ez a megoldás nem biztosítja az
egyedi számlaszámokat

GoodLuck

- <u>lastAccount</u>	: int	= 1000	get
- <u>rate</u>	: double	= 1.3	get/set
- <u>salesCost</u>	: double	= 400.0	get/set
- <u>purchaseCost:double</u>		= 200.0	get/set
- account	: int		get
- name	: string		get
- beneficiary	: string		get/set
- quantity	: int		get
+ <u>newAccount()</u>	: int		
+ <u>howManyUnits(double)</u>	: int		
+ GoodLuck(string, string)			
+ GoodLuck(string)			
+ purchase(double)	: double		
+ sell(double)	: double		
+ value()	: double		
+ toString()	: string		

BadLuck

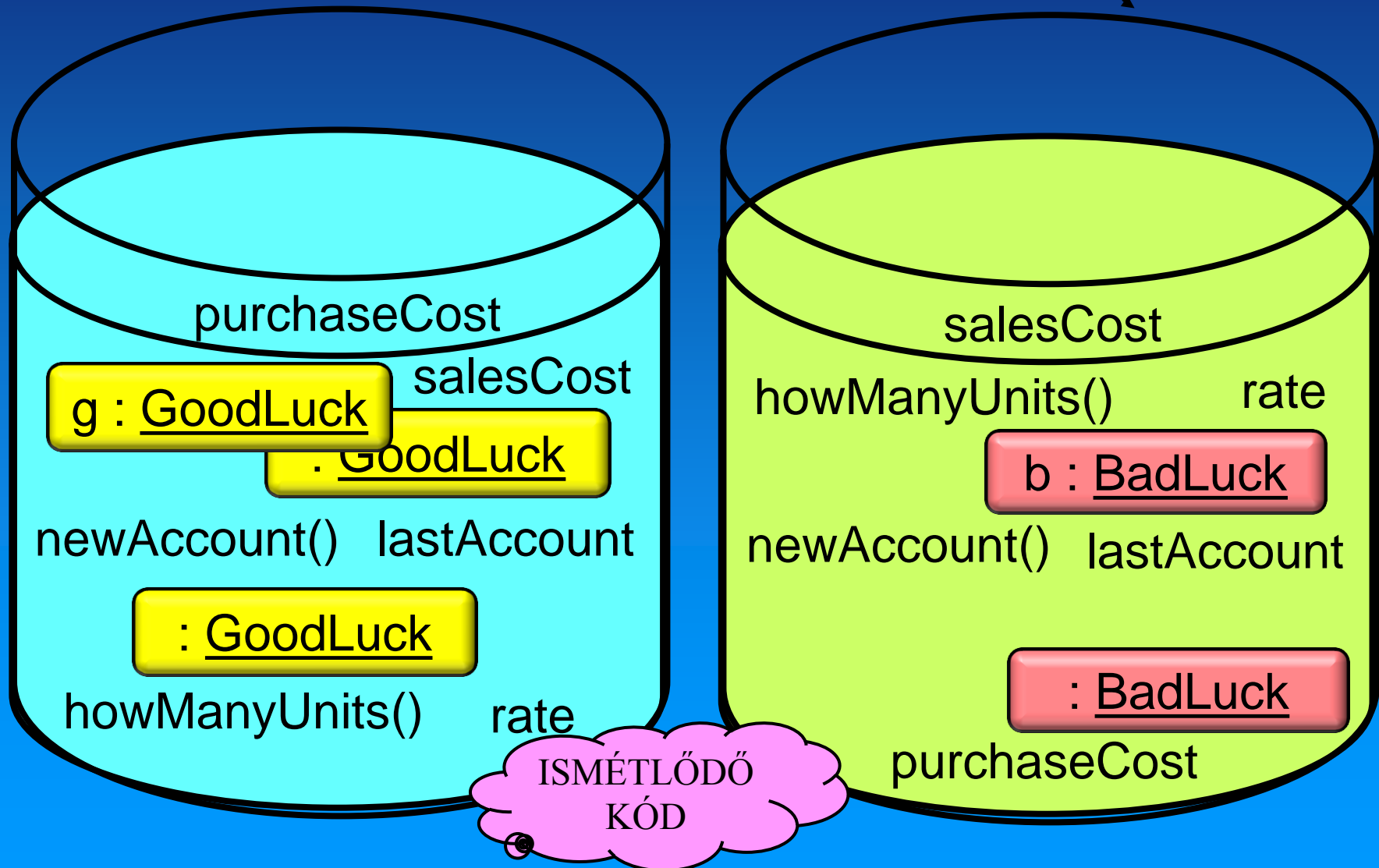
- <u>lastAccount</u>	: int	= 2000	get
- <u>rate</u>	: double	= 1.3	get/set
- <u>salesCost</u>	: double	= 600.0	get/set
- <u>purchaseCost:double</u>		= 300.0	get/set
- account	: int		get
- name	: string		get
- quantity	: int		get
+ <u>newAccount()</u>	: int		
+ <u>howManyUnits(double)</u>	: int		
+ BadLuck(string)			
+ purchase(double)	: double		
+ sell(double)	: double		
+ value()	: double		
+ toString()	: string		

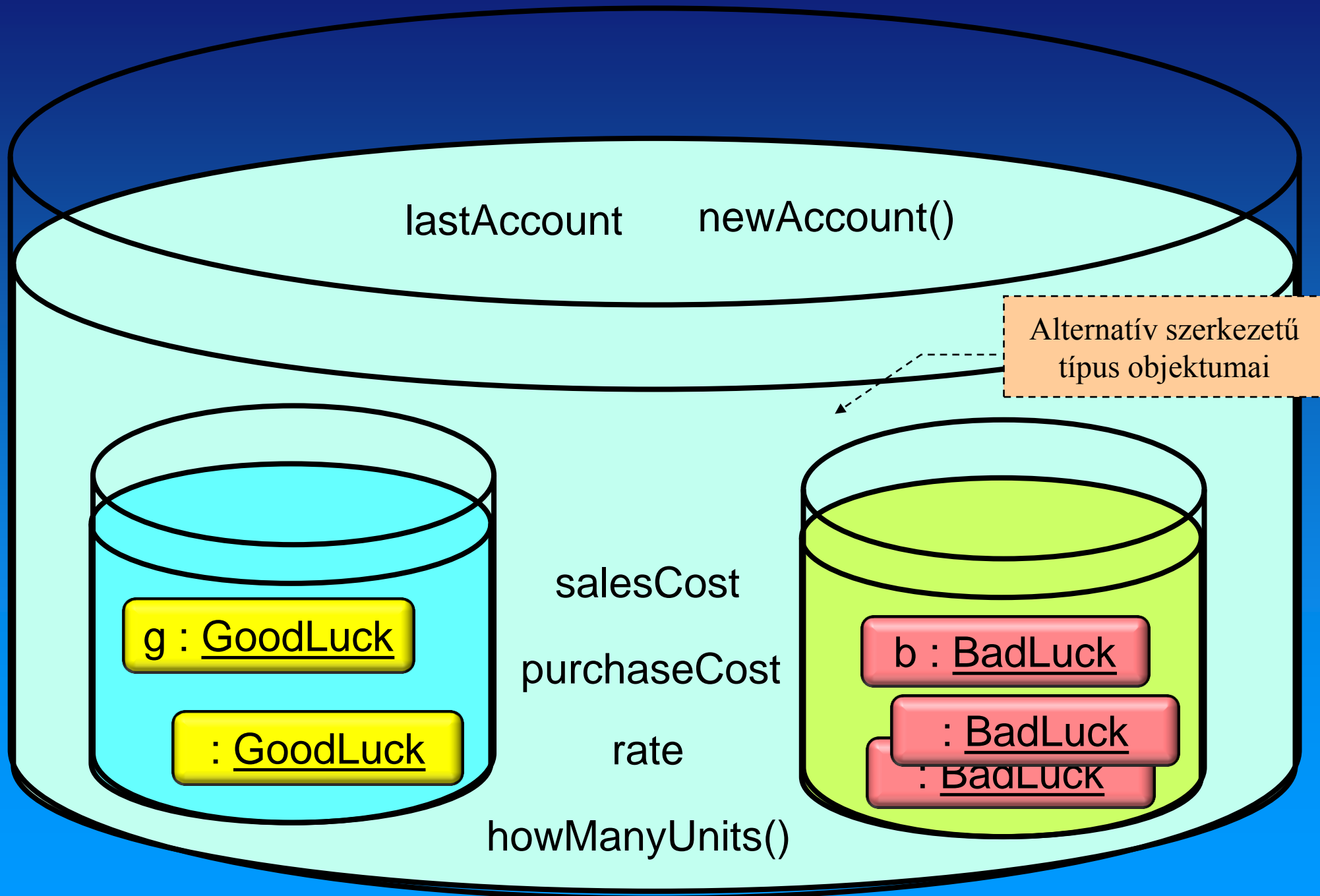
ISMÉTLŐDŐ
KÓD

1. próbálkozás

Objektumdiagram

ez a megoldás nem biztosítja az
egyedi számlaszámokat





2. próbálkozás

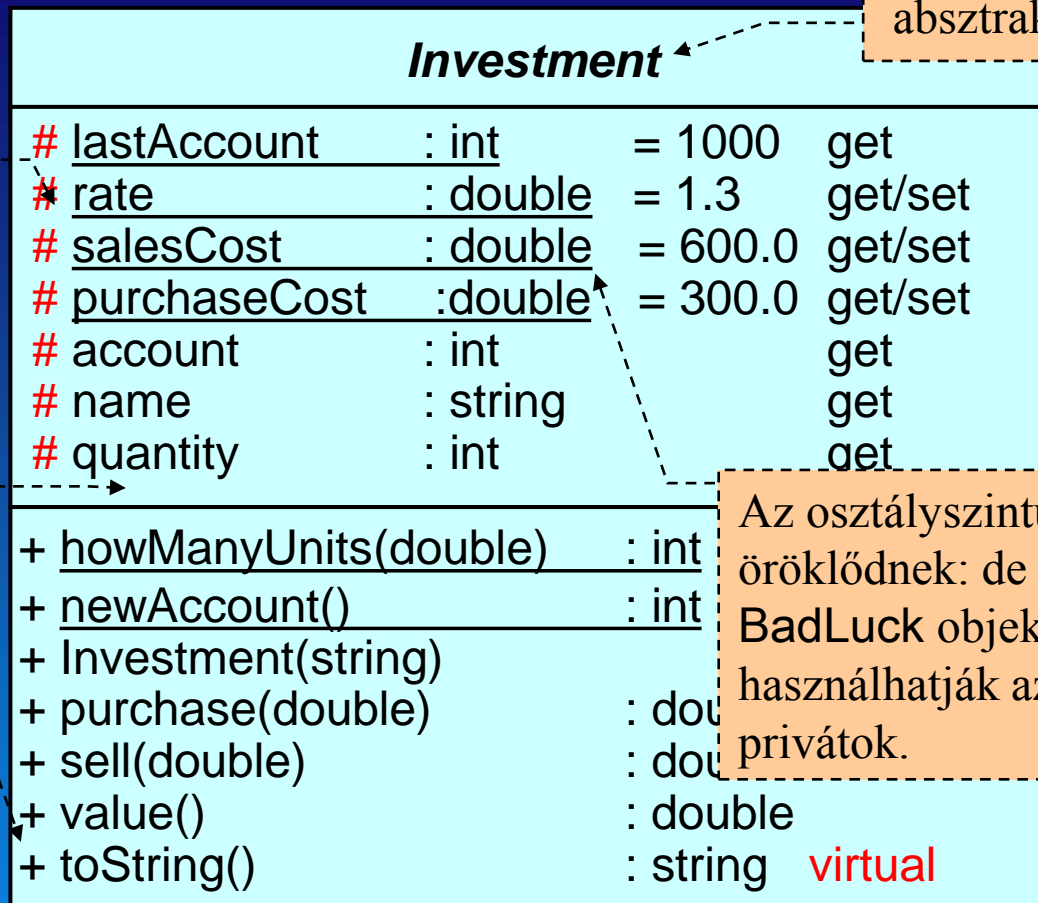
Ez a megoldás nem biztosítja a befektetési jegy fajták szerint egyedi árfolyamát és kezelési költségeit.

Csak publikus (public) és a védett (protected) elemek öröklődnek.

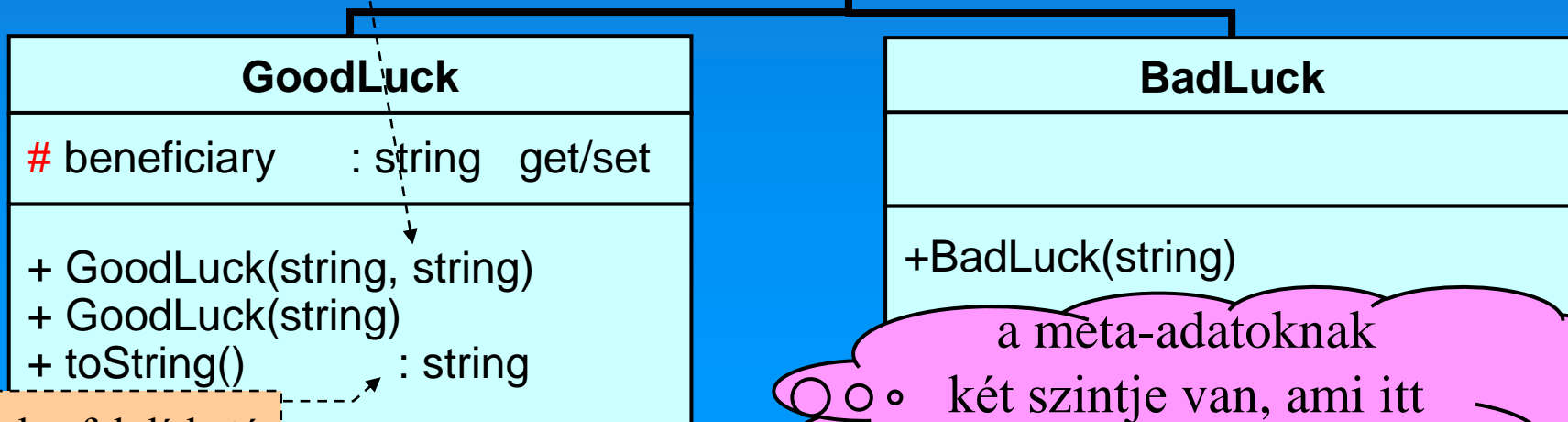
Egyelőre nem definiálunk `is_GoodLuck()` és `is_BadLuck()` szelektorokat

Konstruktor nem öröklődik

Örökölt metódus felülírható



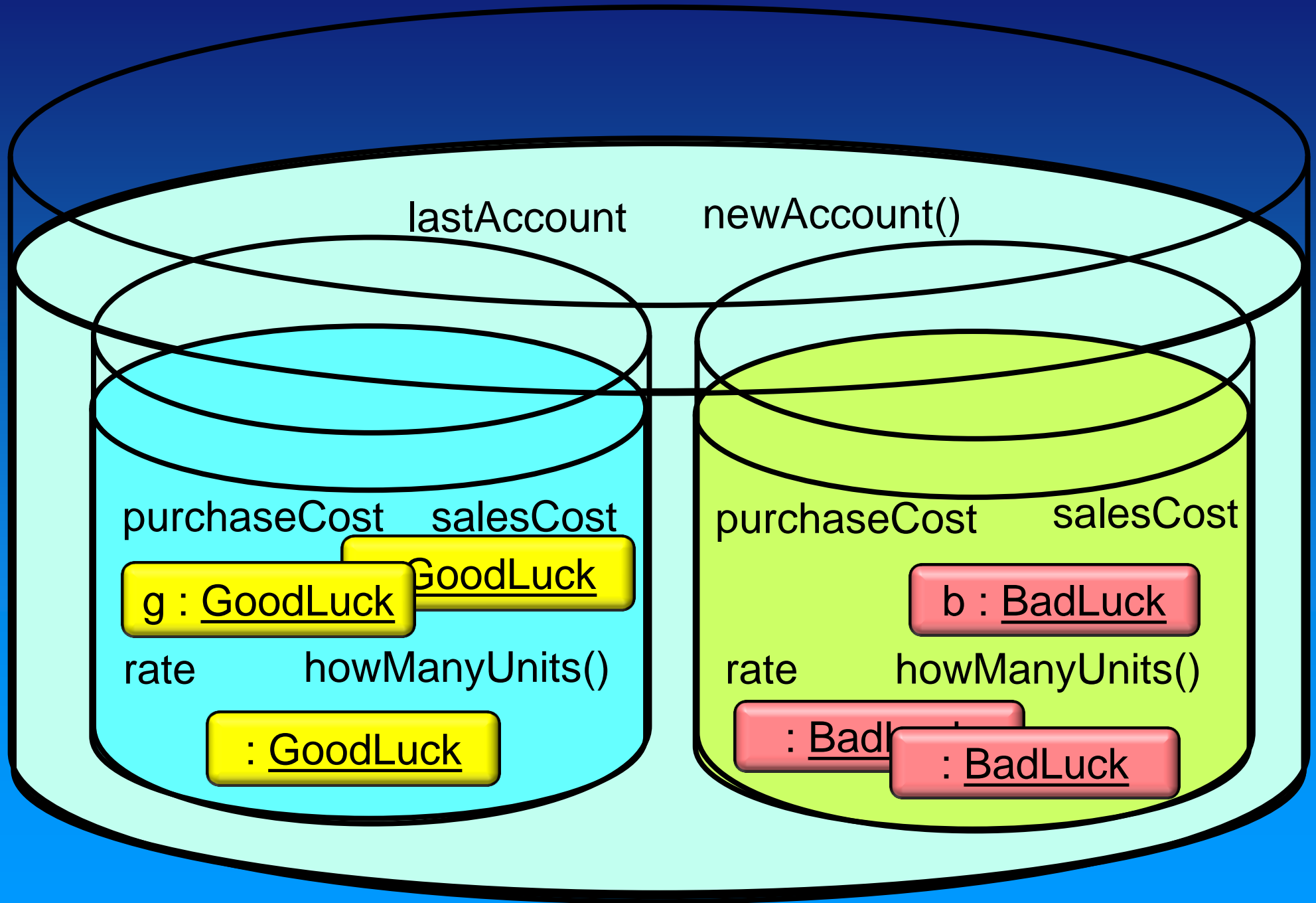
Az osztályszintű elemek nem öröklődnek: de a `GoodLuck` és `BadLuck` objektumok közösen használhatják azokat, ha nem privátok.



○ ○ ○ a meta-adatoknak két szintje van, ami itt nem jelenik meg

3. próbálkozás

Objektumdiagram



3. próbálkozás

<i>Investment</i>			
# <u>lastAccount</u>	: int	= 1000	get
# <u>account</u>	: int		get
# <u>name</u>	: string		get
# <u>quantity</u>	: int		get
<hr/>			
+ <u>newAccount()</u>	: int		
+ <u>Investment(string)</u>			
+ <u>toString()</u>	: string	virtual	
+ <u>purchase(double)</u>	: double	abstract	
+ <u>sell(double)</u>	: double	abstract	
+ <u>value()</u>	: double	abstract	

Érdemes bevezetni ezeket a deklarációkat, kihasználhassuk a dinamikus kötést

```
Investment *o  
    = new BadLuck(...);  
double d = o ->value();
```



Jól elkülönül a meta-adatok két szintje

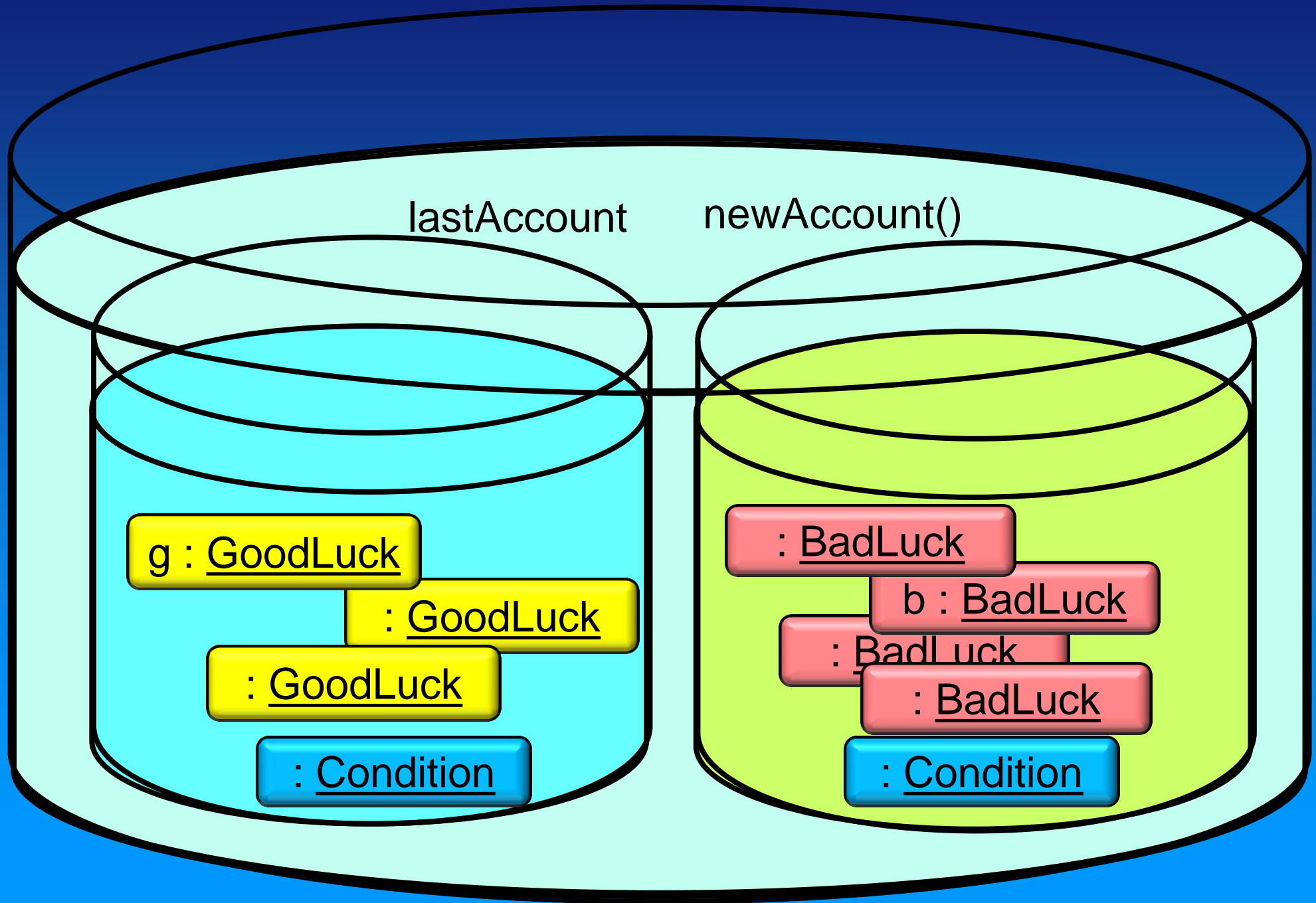
GoodLuck			
# <u>rate</u>	: double	get/set	
# <u>salesCost</u>	: double	get/set	
# <u>purchaseCost</u>	: double	get/set	
# <u>beneficiary</u>	: string	get/set	
<hr/>			
+ <u>howManyUnits(a : double)</u>	: int		
+ <u>GoodLuck(string, string)</u>			
+ <u>GoodLuck(string)</u>			
+ <u>purchase(double)</u>	: double		
+ <u>sell(double)</u>	: double		
+ <u>value()</u>	: double		
+ <u>toString()</u>	: string		

BadLuck			
# <u>rate</u>	: double	get/set	
# <u>salesCost</u>	: double	get/set	
# <u>purchaseCost</u>	: double	get/set	
<hr/>			
+ <u>howManyUnits(a : double)</u>	: int		
+ <u>BadLuck(string)</u>			
+ <u>purchase(double)</u>	: double		
+ <u>sell(double)</u>	: double		
+ <u>value()</u>	: double		

ISMÉTLŐDŐ
KÓD

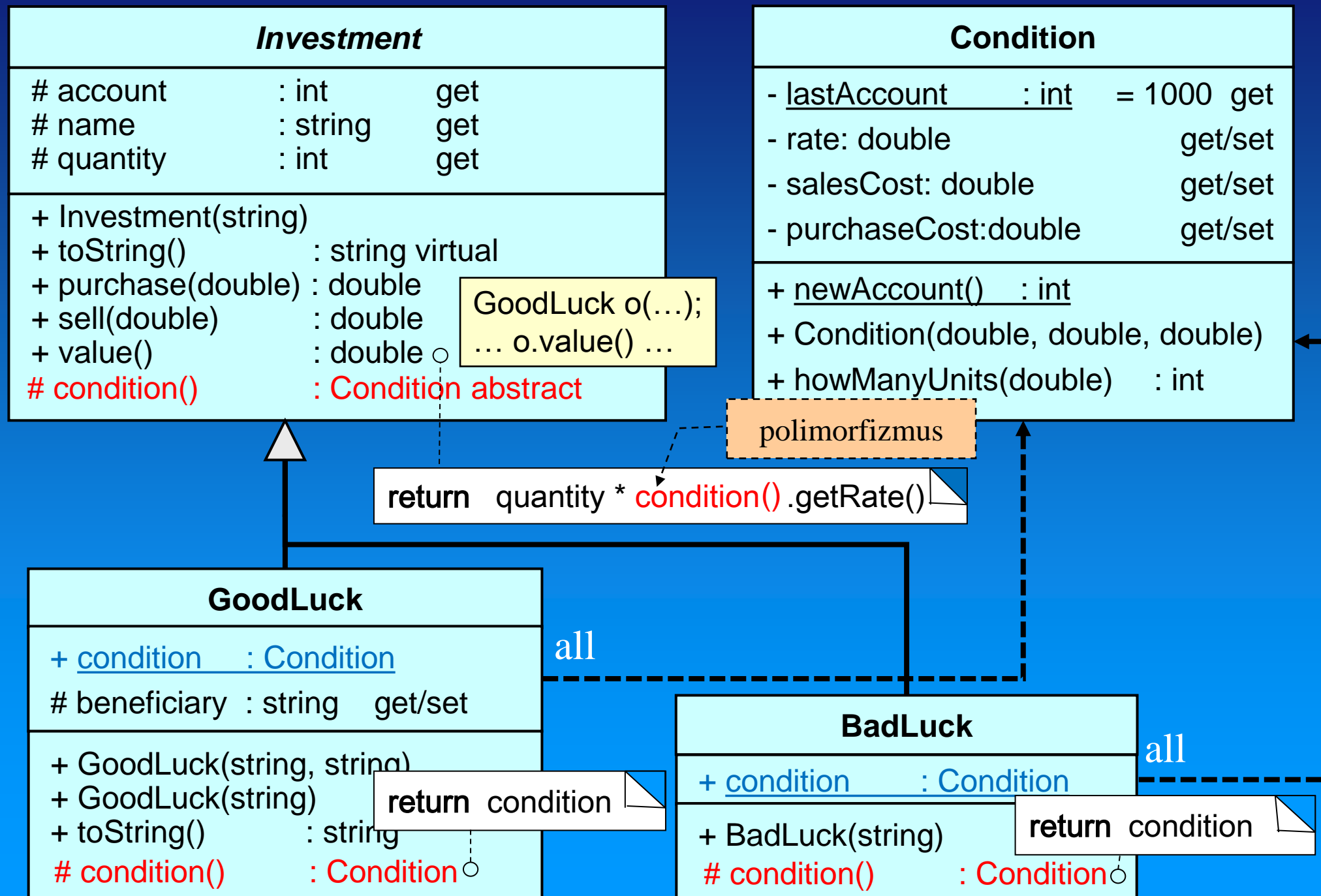
4. próbálkozás

Objektumdiagram



4. próbálkozás

○ ○ ○ Jól elkülönül a meta-adatok két szintje




```

class Condition{
private:
    // osztály adattag
    static int _lastAccount;
    // példány adattagok
    double _rate;
    double _salesCost;
    double _purchaseCost;

public:
    // osztálymetódusok
    static int newAccount() { return ++_lastAccount; }
    // konstruktor
    Condition(double a, double b, double c)
        : _rate(a), _salesCost(b), _purchaseCost(c) {}
    // példánymetódus
    int howManyUnits(double amount) const { return (int)amount/_rate; }
    // hozzáférés az osztály adattaghoz
    static int getLastAccount() const { return _account; }
    // hozzáférés az adattagokhoz
    double getRate() const { return _rate; }
    void setRate(double c) { _rate = c; }
    double getSalesCost() const { return _salesCost; }
    void setSalesCost(double c) { _salesCost = c; }
    double getPurchaseCost() const { return _purchaseCost; }
    void setPurchaseCost(double c) { _purchaseCost
};

```

```

// osztály adattag
int Condition::_lastAccount = 1000;

```

condition.cpp

condition.h

```

class Investment
{
protected:
    // adattagok
    int          _account;
    std::string  _name;
    int          _quantity;

    virtual Condition condition() const = 0;

    // konstruktor
    Investment(const std::string &str);

public:
    // példánymetódusok
    double value() const;
    double purchase(double amount);
    double sell(double amount);
    virtual std::string toString() const;

    // hozzáférés az adattagokhoz
    int getAccount()      const { return _account; }
    std::string getName() const { return _name; }
    int getQuantity()    const { return _quantity; }
};

```

a származtatott osztályok kondícióját
tartalmazó statikus tag lekérdezésének
absztrakt példánymetódusa

absztrakt osztály:

- van absztrakt metódusa
- nincs publikus konstruktora

// konstruktor

```
Investment::Investment(const std::string &str)
{
    _account = Condition::newAccount();
    _name = str;
    _quantity = 0;
}
```

// példánymetódusok

```
double Investment::value() const
{
    return _quantity*condition().getRate();
}
```

egy konkrét GoodLuck vagy BadLuck objektumra meghívva a value() metódust, ez a megfelelő condition() metódust használja majd.

```
std::string Investment::toString() const
{
    std::ostringstream ss;
    ss << "Számola: " << _account <<
        "Név: " << _name <<
        "jegyek száma: " << _quantity;
    return ss.str();
}
```

```
double Investment::sell(double amount)
{
    int db = condition().howManyUnits(
        amount + condition().getSalesCost());
    if(_quantity >= db) _quantity -= db;
    else { db = _quantity; _quantity = 0; }
    return db * condition().getRate() - condition().getSalesCost();
}
```

egy konkrét GoodLuck vagy BadLuck objektumra meghívva a `sell()` metódust, ez a megfelelő `condition()` metódust használja

```
double Investment::purchase(double amount)
{
    int db = condition().howManyUnits(
        amount - condition().getPurchaseCost());
    _quantity += db;
    return db * condition().getRate() + condition().getPurchaseCost();
}
```

egy konkrét GoodLuck vagy BadLuck objektumra meghívva a `purchase()` metódust, ez a megfelelő `condition()` metódust használja majd.

```

class GoodLuck : public Investment
{
public:
    // osztály adata
    static Condition _condition;
protected:
    // példány adata
    std::string _beneficiary;

    // példánymetódus
    Condition condition() const { return _condition; }
public:
    // konstruktorok
    GoodLuck(const std::string &str1, const std::string &str2);
    GoodLuck(const std::string &str);

    // példánymetódusok
    std::string toString() const;
    std::string getBeneficiary() const { return _beneficiary; }
    void setBeneficiary(std::string str) { _beneficiary = str; }
};

```

GoodLuck osztályhoz rendelt
statikus Condition objektum

a _condition objektumot
visszaadó példánymetódus

```

// osztály adattag
Condition GoodLuck::_condition(4.8, 400, 200);

// konstruktorok
GoodLuck::GoodLuck(const std::string &str) : Investment(str)
{
    _beneficiary = "";
}

GoodLuck::GoodLuck(const std::string &str1, const std::string &str2)
: Investment(str1)
{
    _beneficiary = str2;
}

// példánymetódusok
std::string GoodLuck::toString() const
{
    std::ostringstream ss;
    ss << "Számla: " << _account <<
        "Név: " << _name <<
        "Kedvezményezett: " << _beneficiary <<
        "jegyek száma: " << _quantity;
    return ss.str();
}

```

GoodLuck osztályhoz rendelt
statikus Condition objektum létrehozása

```

class BadLuck : public Investment
{
public:
    // osztály adattag
    static Condition _condition;
protected:
    // példány metódus
    Condition condition() const { return _condition; }
public:
    // konstruktor
    BadLuck(const std::string &str) : Investment(str) {}
};

```

BadLuck osztályhoz rendelt
statikus Condition objektum

a Condition objektumot visszaadó
példánymetódus felüldefiniálása

badluck.h

```

// osztály adattag
Condition BadLuck::_condition(2.4, 600, 300);

```

BadLuck osztályhoz rendelt
statikus Condition objektum

badluck.cpp

```
int main()
{
    GoodLuck g("Bólyai Farkas", "Bólyai János");
    BadLuck b("Riesz Frigyes");

    cout << "GoodLuck árfolyam: " <<
        GoodLuck::_condition.getRate() << endl;
    cout << "BadLuck árfolyam: " <<
        BadLuck::_condition.getRate() << endl;

    cout << "Vásárol: " << g.purchase(5000) << endl;
    cout << "Vásárol: " << b.purchase(5000) << endl << endl;

    cout << g.toString() << endl << b.toString() << endl;

    GoodLuck::_condition.setRate(8.9);
    BadLuck::_condition.setRate(1.2);

    cout << "Elad: " << g.sell(5000) << endl;
    cout << "Elad: " << b.sell(2000) << endl;

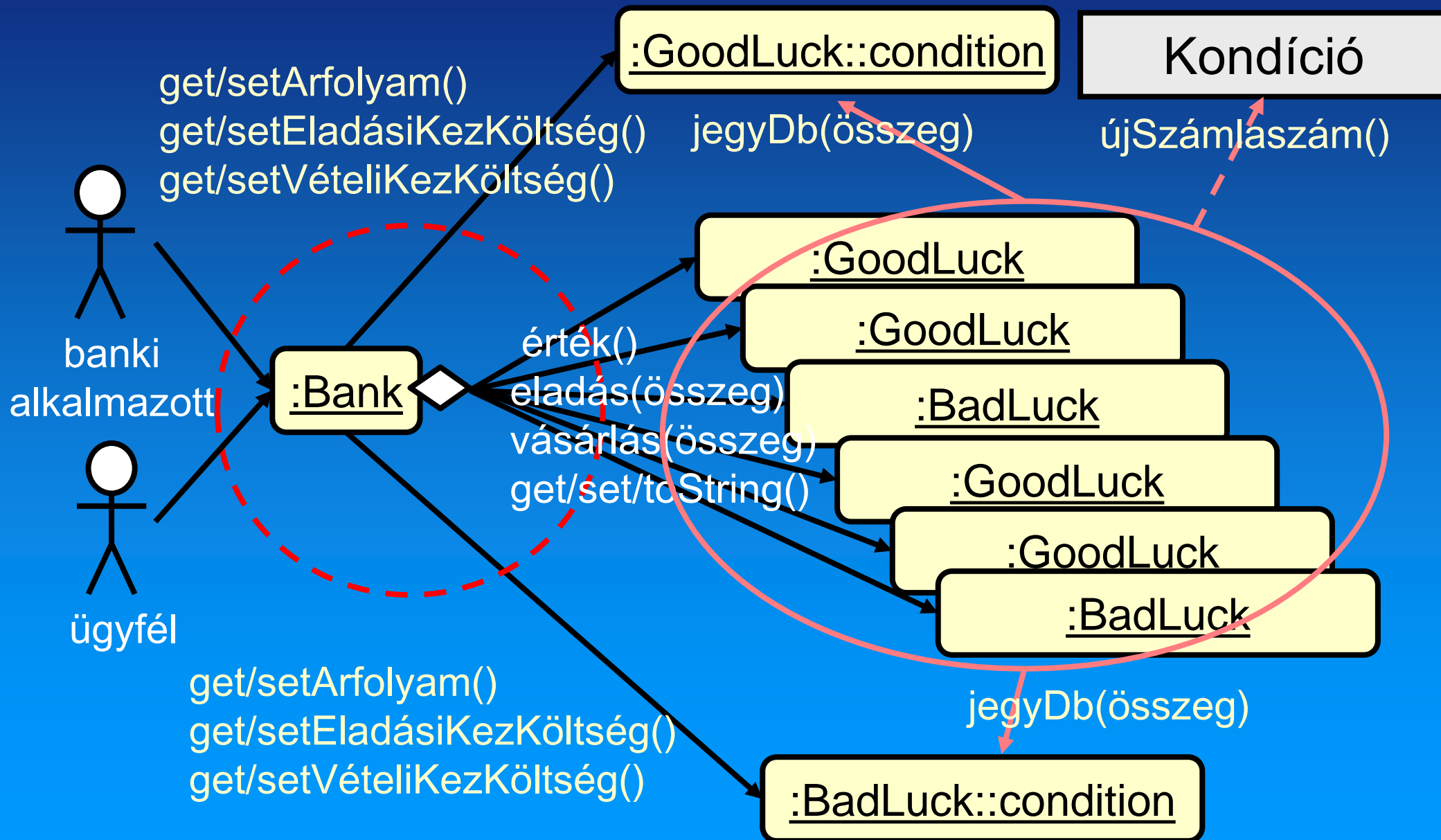
    cout << g.toString() << endl << b.toString() << endl;

    return 0;
}
```

árfolyam lekérdezés

árfolyam változtatás

Együttműködési diagram



Objektum orientáltság ismérvei

1. Egységbe zárás

- Az osztály a korszerű típus fogalom leírásának eszköze: együtt jelennek meg benne a típus egy értékét (objektumát) leíró (reprezentáló) **adattagok** és egy értékkel (objektummal) műveletet végző **metódusok**.
- Az osztály mindegyik példánya (objektuma) rendelkezik az osztályban definiált adattagokkal és metódusokkal.

2. Elrejtés

- Az osztály, miközben az általa leírt típus interfészét megmutatja (a típus metódusai publikusak) az implementációját elrejtí (a reprezentáció privát).
- Az osztály objektumainak adattagjai és metódusai globálisak az objektum metódusaira nézve, de más programrészekben csak akkor láthatóak, ha ezt külön engedélyezzük (public, protected).

Objektum orientáltság ismérvei (folyt.)

3. Öröklődés

- Kódismétlődés (redundancia) megszüntetésének eszköze, amely segítségével egy osztályt egy másik osztályhoz hasonlóan definiálhatunk: egy osztály örökölheti egy másik (az őssztály) tulajdonságait (adattagjait és metódusait)
- Felhasználásának néhány jellegzetes módjai:
 - **Származtatott típus** osztálya újabb tulajdonságokat vehet hozzá az őssztályhoz, az öröklött metódusok definícióját módosíthatja.
 - **Altípus** osztálya korlátozhatja némelyik öröklött adattag lehetséges értékeit, illetve letilthatja némelyik öröklött metódus használatát.
 - **Alternatív szerkezetű típus** megvalósítása (szelektor függvényekkel)

Objektum orientáltság ismérvei (folyt.)

4. Többalakúság (polimorfizmus)

- Az a jelenség, amikor egy öröklődési láncban újra és újra felüldefiniált metódusnak mindig azon osztálybeli **változata hajtódik végre**, amelyiknek egy objektumára a metódust meghívjuk.
- Ez az elv áttételesen is érvényesül. Amikor egy objektumra meghívunk egy ősosztályból származó olyan *A* metódust, amelyet nem definiáltunk felül, és amelyik meghív egy olyan *B* metódust, amelyet nemcsak az ősosztály definiál, hanem az objektum osztálya is felüldefiniálja, akkor az *A* végrehajtása során a *B* metódus felüldefiniált változata hajtódik majd végre.

Objektum orientáltság ismérvei (folyt.)

5. Dinamikus kötés

- Amikor egy őszosztály típusú (C++ esetén pointer) **változónak értékül adunk egy származtatott osztályú objektumot** (C++-ban a címét), és erre a változóra **meghívjuk az őszosztály egy virtuális metódusát**, akkor ennek a **származtatott osztályban felüldefiniált változata fut le** feltéve, hogy van ilyen.
- Fordítási időben, amikor egy ilyen változónak csak a típusát ismerjük (ez most az őszosztály), de az értékét nem: a változóval csak az őszosztály metódusai kapcsolhatóak össze. Az majd csak a végrehajtás során derül ki, hogy a változó valójában milyen típusú objektumot rejt, azaz valójában melyik metódust kell a változóhoz kötni, amikor a változóval meghívjuk azt.