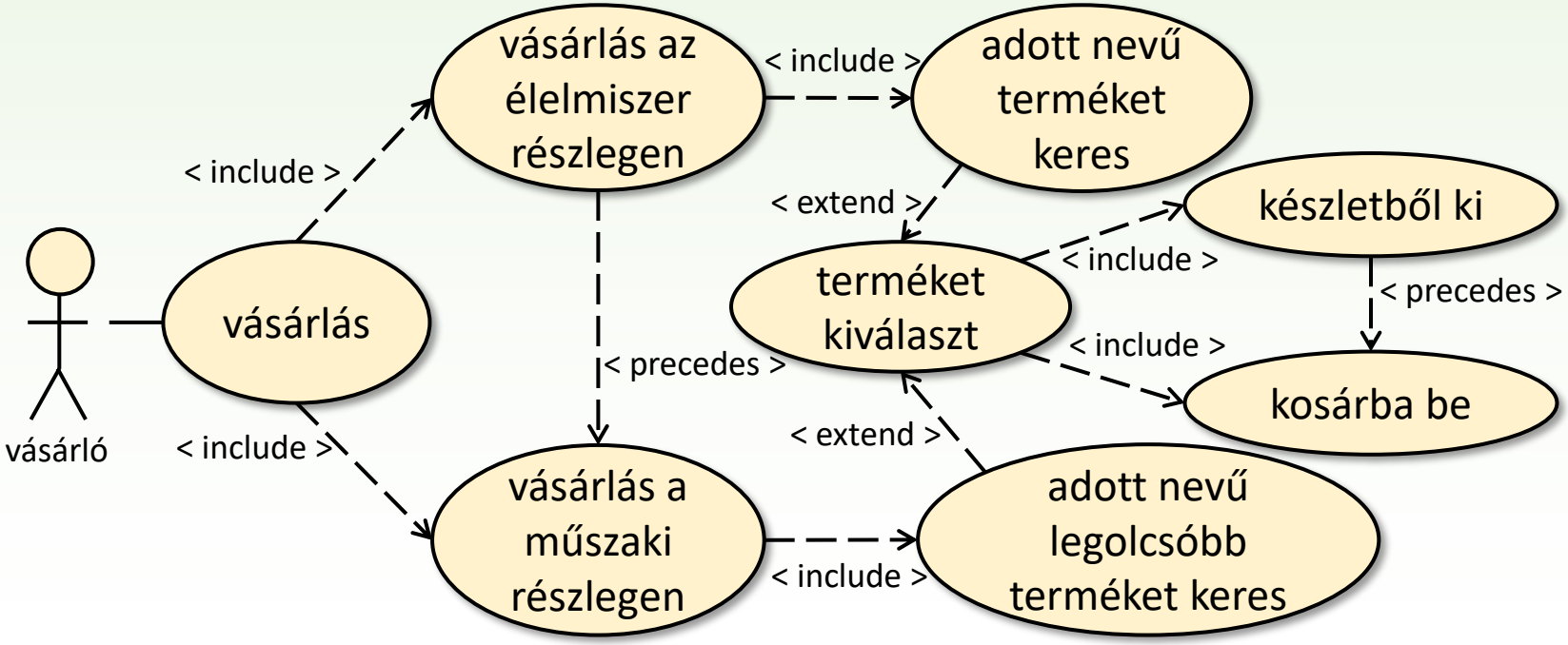
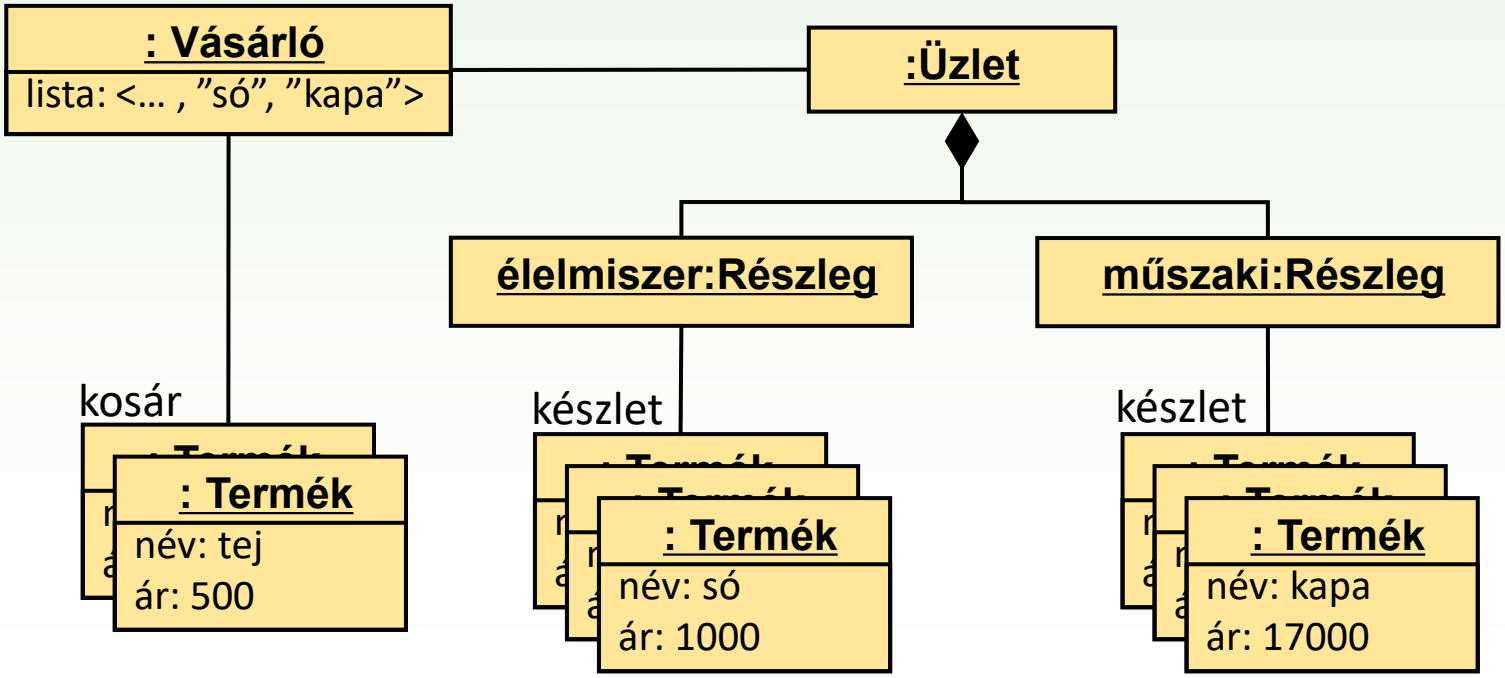
Vásárlás

Egy kisvárosi üzlet élelmiszer részlegből és műszaki részlegből áll, ahová a vásárlók egy bevásárlólistával jönnek, amely azon termékek neveit tartalmazza, amit megvennének. Az üzletben a listájukon szereplő termékeket keresik: először az élelmiszer részlegen nézik végig a teljes bevásárlólistát, és a megtalált termékeket magukhoz veszik (beteszik a kosarukba), majd a műszaki részlegen ezt megismétlik, de megfontoltabban: ha egy (a bevásárlólistán szereplő) termékből több is van a részlegen, akkor a legolcsóbbat választják.



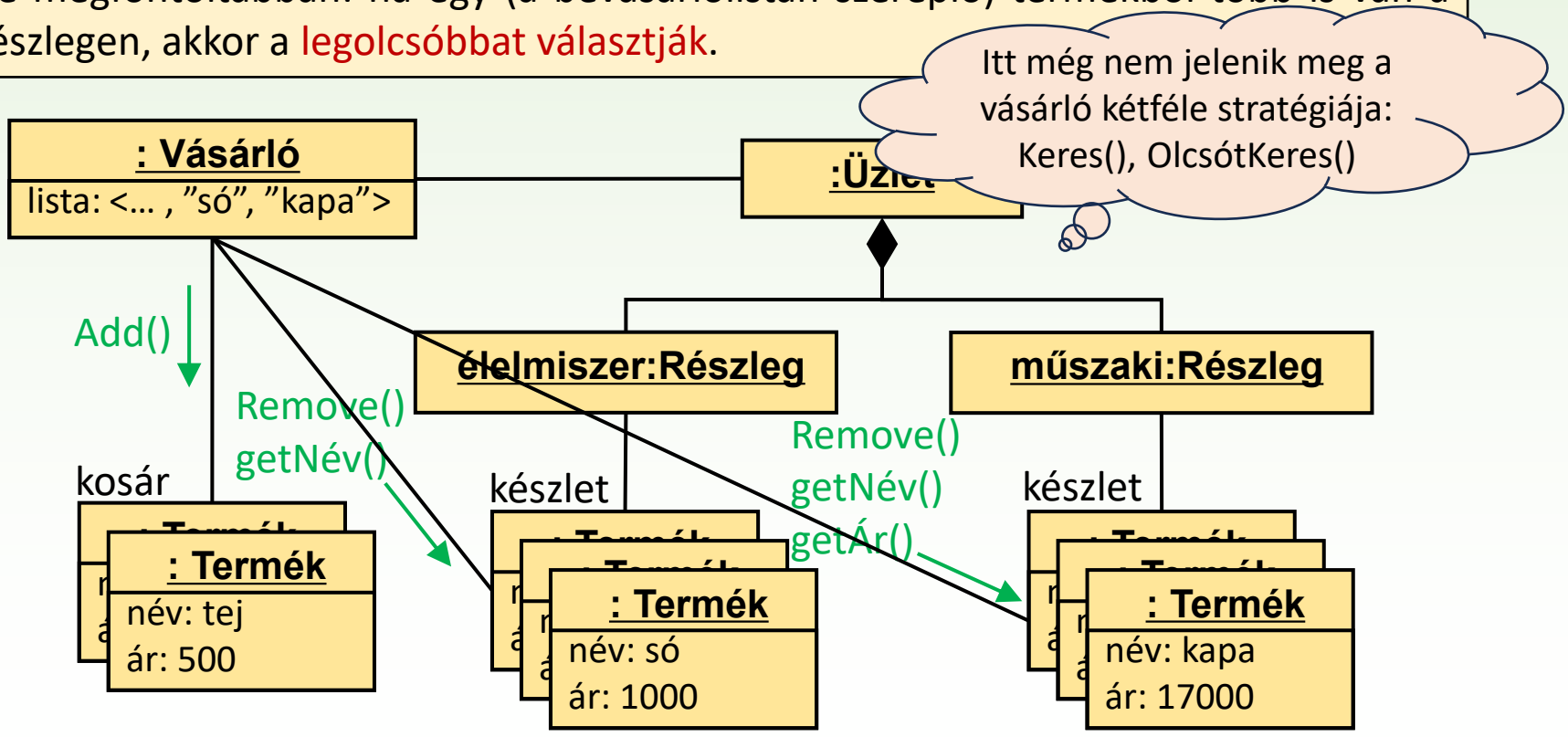
Vásárlás

Egy kisvárosi üzlet élelmiszer részlegből és műszaki részlegből áll, ahová a vásárlók egy bevásárlólistával jönnek, amely azon termékek neveit tartalmazza, amit megvennének. Az üzletben a listájukon szereplő termékeket keresik: először az élelmiszer részlegen nézik végig a teljes bevásárlólistát, és a megtalált termékeket magukhoz veszik (beteszik a kosarukba), majd a műszaki részlegen ezt megismétlik, de megfontoltabban: ha egy (a bevásárlólistán szereplő) termékből több is van a részlegen, akkor a legolcsóbbat választják.

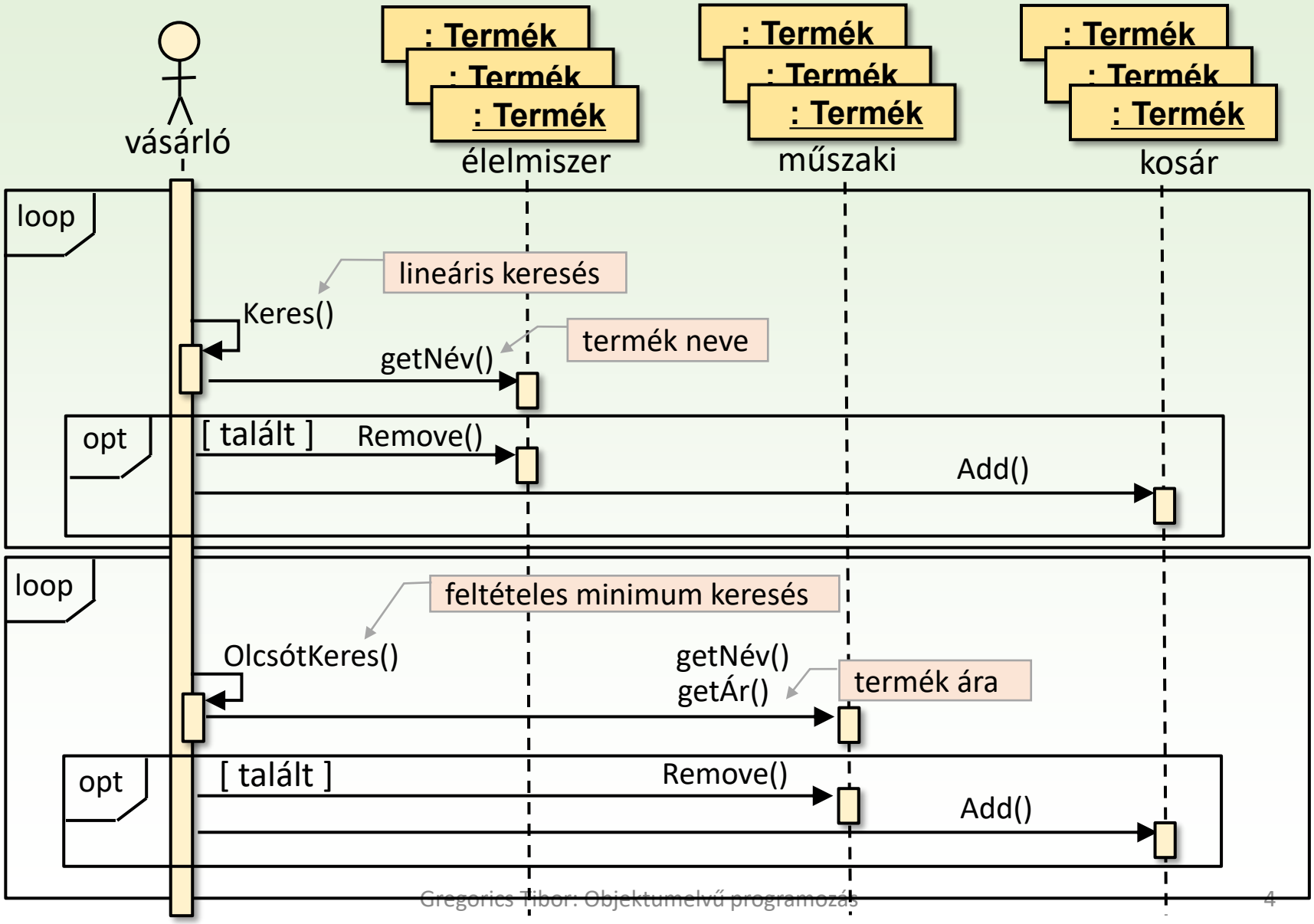


Vásárlás

Egy kisvárosi üzlet élelmiszer részlegből és műszaki részlegből áll, ahová a vásárlók egy bevásárlólistával jönnek, amely azon termékek neveit tartalmazza, amit megvennének. Az üzletben a listájukon szereplő termékeket keresik: először az élelmiszer részlegen nézik végig a teljes bevásárlólistát, és a megtalált termékeket magukhoz veszik (beteszik a kosarukba), majd a műszaki részlegen ezt megismétlik, de megfontoltabban: ha egy (a bevásárlólistán szereplő) termékből több is van a részlegen, akkor a legolcsóbbat választják.

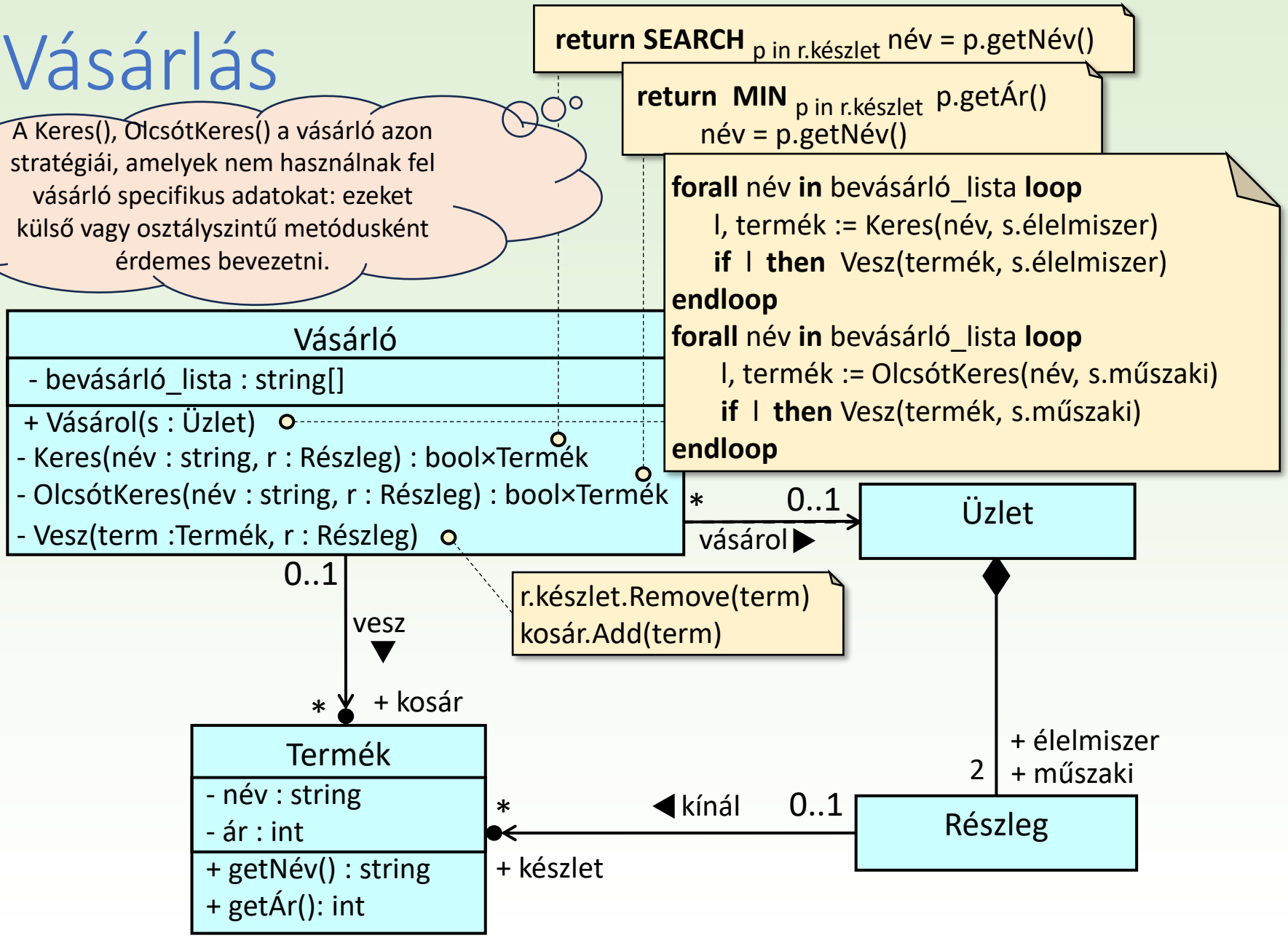


Vásárlás


vásárló
: Termék
: Termék
: Termék
élelmiszer
: Termék
: Termék
: Termék
műszaki
: Termék
: Termék
: Termék
kosár
loop
lineáris keresés
Keres()
termék neve
getNév()
opt
[talált]
Remove()
Add()
loop
feltételes minimum keresés
OlcsótKeres()
getNév()
getÁr()
termék ára
opt
[talált]
Remove()
Add()

Vásárlás

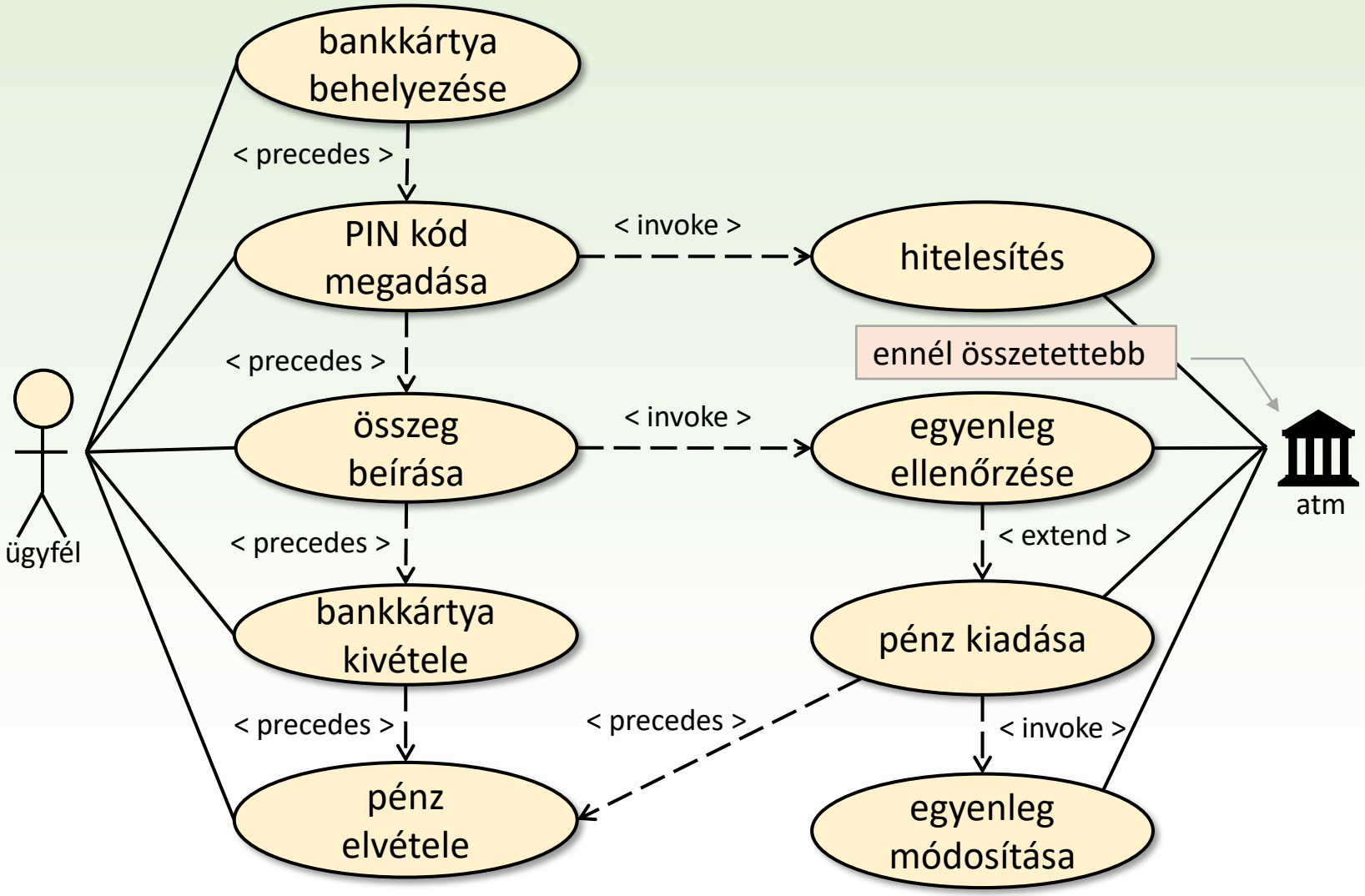
A Keres(), OlcsótKeres() a vásárló azon stratégiái, amelyek nem használnak fel vásárló specifikus adatokat: ezeket külső vagy osztályszintű metódusként érdemes bevezetni.



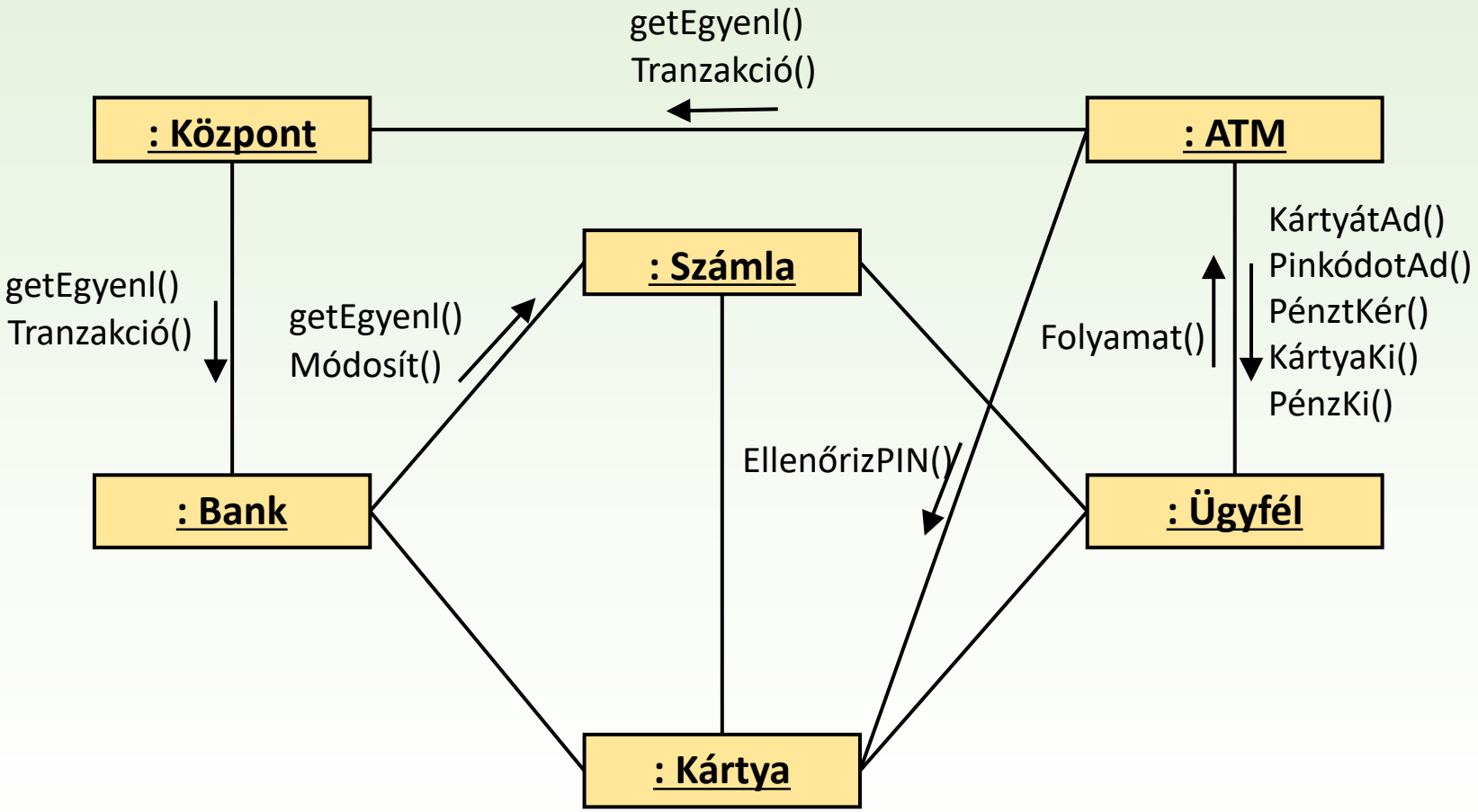
Pénzfelvétel

Egy ATM automatánál az ügyfelek sorban állnak, hogy pénzt vehessenek fel. Az ügyfelek rendelkeznek bankkártyákkal. Egy bankkártya egy bankszámlához tartozik, és van egy PIN kódja. Egy ügyfél odaadja a bankkártyáját és a PIN kódját az ATM-nek, és az ellenőrzi ennek hitelességét. Ezután az ügyfél megadja a felvenni kívánt összeget. Ha az összeget levonva az ügyfél számlájának egyenlegéből az továbbra is pozitív marad, akkor az ATM kiadja az összeget. Ehhez a folyamathoz az ATM egy központon keresztül lekéri az ügyfél számlaegyenlegét a kártyájának adatai alapján, illetve elküld a egy jelentést a lebonyolított tranzakcióról az ügyfél bankjának, amely ezalapján leveszi az összeget az ügyfél számlájáról.

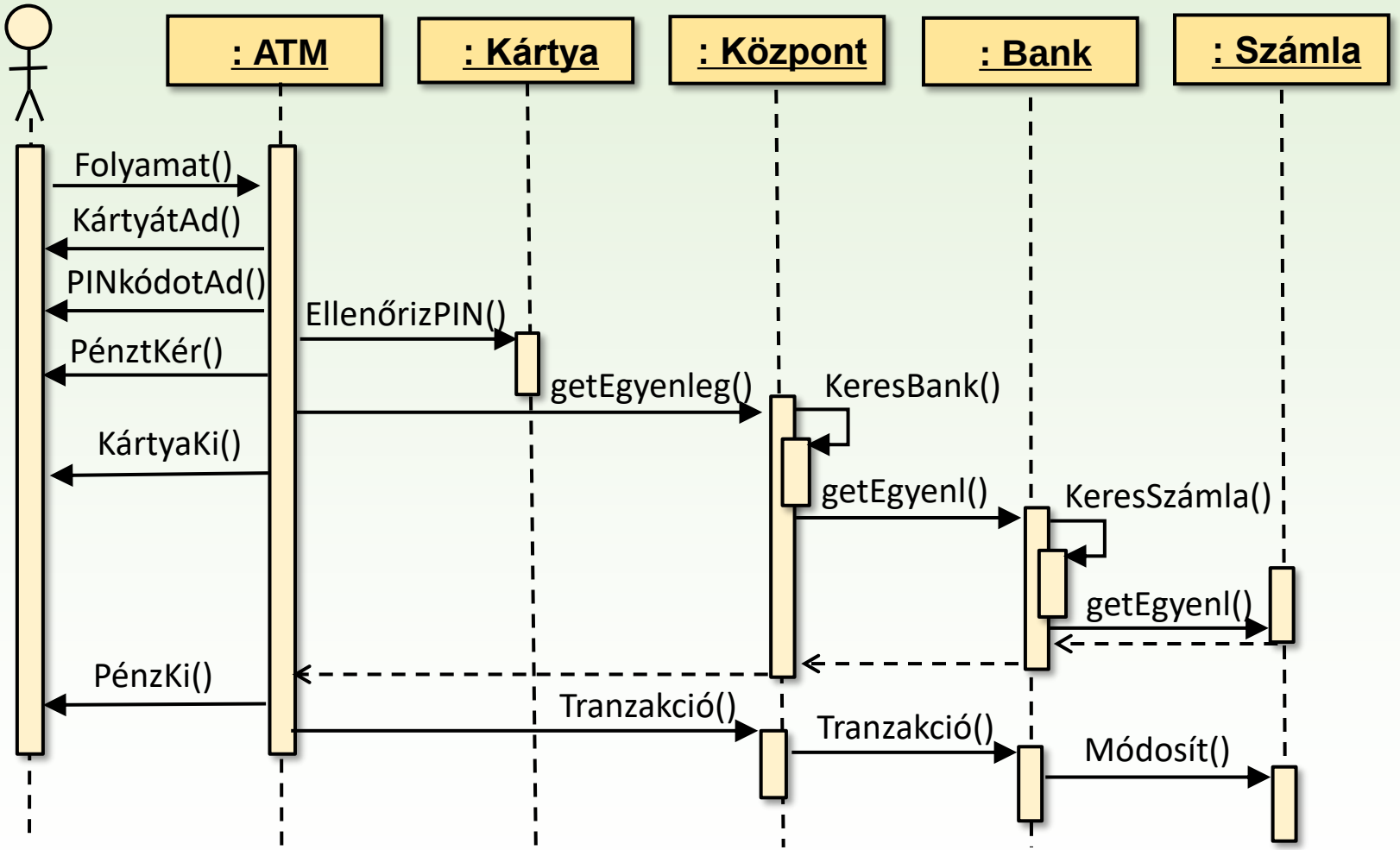
Pénzfelvétel



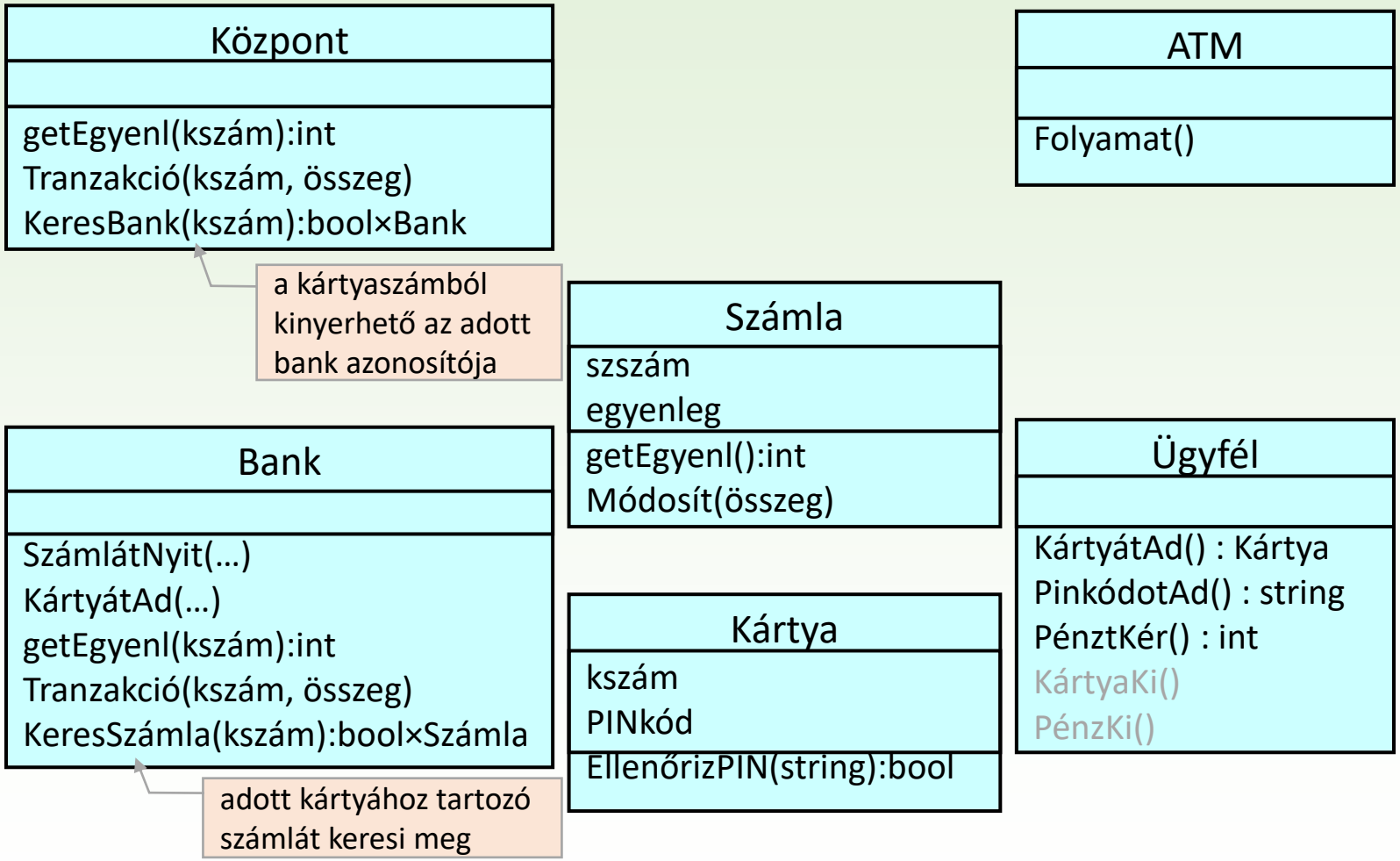
Pénzfelvétel



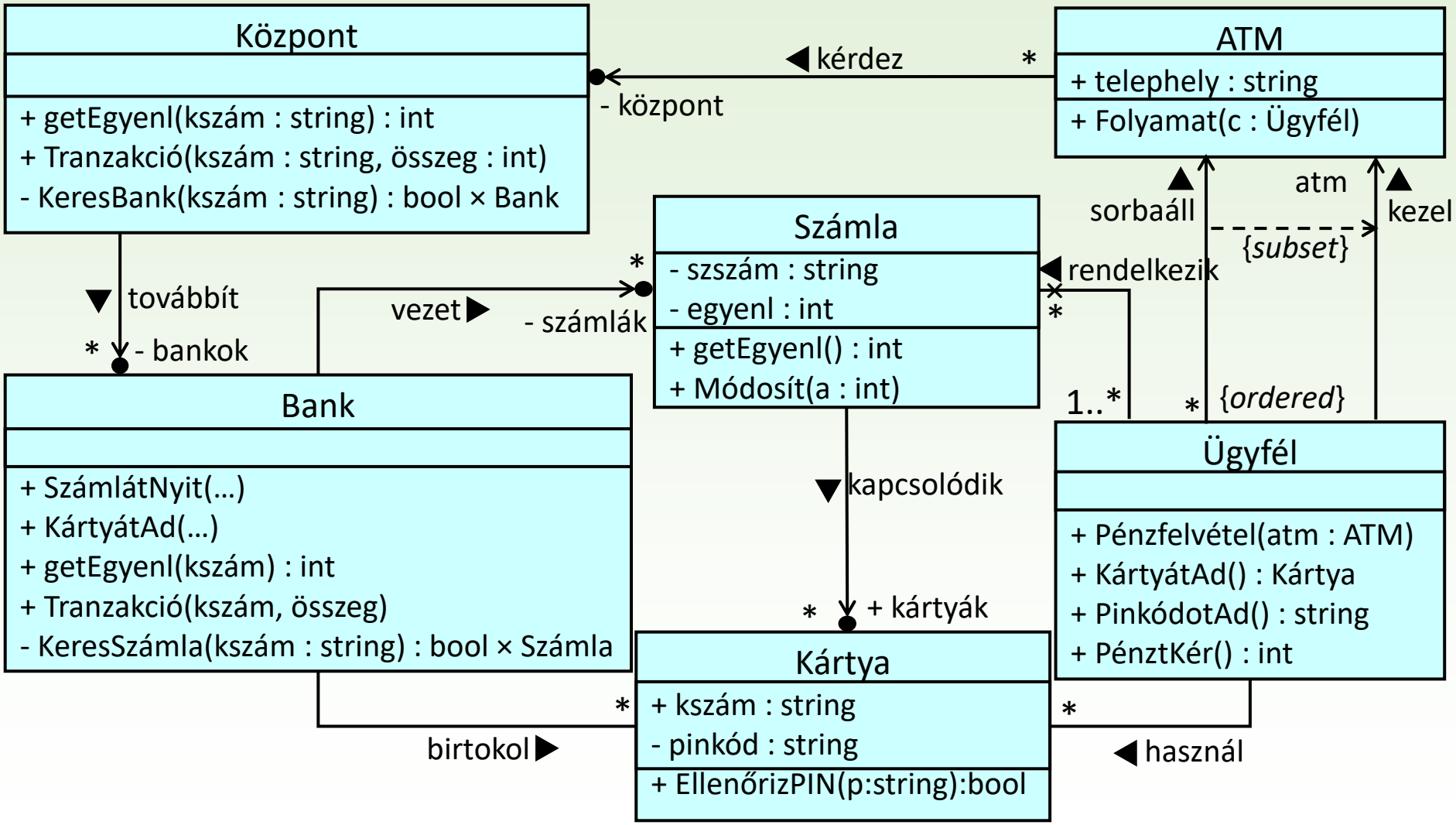
Pénzfelvétel



Pénzfelvétel



Pénzfelvétel



KVÍZ

```
l, bank := KeresBank(kszám)
if l then bank.Tranzakció(kszám, összeg)
```

```
kártya := c.KártyátAd()
if kártya.EllenőrizPIN(c.PinkódotAd()) then
   a := c.PénztKér()
    if központ.getEgyenl(kártya.kszám)>a then
      központ.Tranzakció(kártya.kszám, - a) // pénzt kiad
   endif
endif
```

```
l, bank := KeresBank(kszám)
if l then return bank.getEgyenl(kszám)
```

Központ

+ getEgyenl(kszám : string) : int
+ Tranzakció(kszám : string, összeg : int)
- KeresBank(kszám : string) : bool × Bank

kérdez * — 1 - központ

ATM

+ telephely : string
+ Folyamat(c : Ügyfél)

return egyenl

továbbít — * - bankok

vezet — * - számlák

Számla

+ szszám : string
- egyenl : int
+ getEgyenl() : int
+ Módosít(a : int)

sorbaáll — atm {subset} — kezel

rendelkezik * — 1..* {ordered}

Bank

+ SzámlátNyit(…)
+ KártyátAd(…)
+ getEgyenl(kszám) : int
+ Tranzakció(kszám, összeg)
- KeresSzámla(kszám : string) : bool × Számla

egyenl := egyenl + a

kapcsolódik

return pinkód=p

* + kártyák

Ügyfél

+ Pénzfelvétel(atm : ATM)
+ KártyátAd() : Kártya
+ PinkódotAd() : string
+ PénztKér() : int

```
l, számla := KeresSzámla(kszám)
if l return számla.getEgyenl()
```

```
l, számla := KeresSzámla(kszám)
if l return számla.Módosít(összeg)
```

birtokol * — 1

Kártya

+ kszám : string
- pinkód : string
+ EllenőrizPIN(p:string):boo

használ * — 1

```
// sorbaáll
atm.Folyamat(this)
```

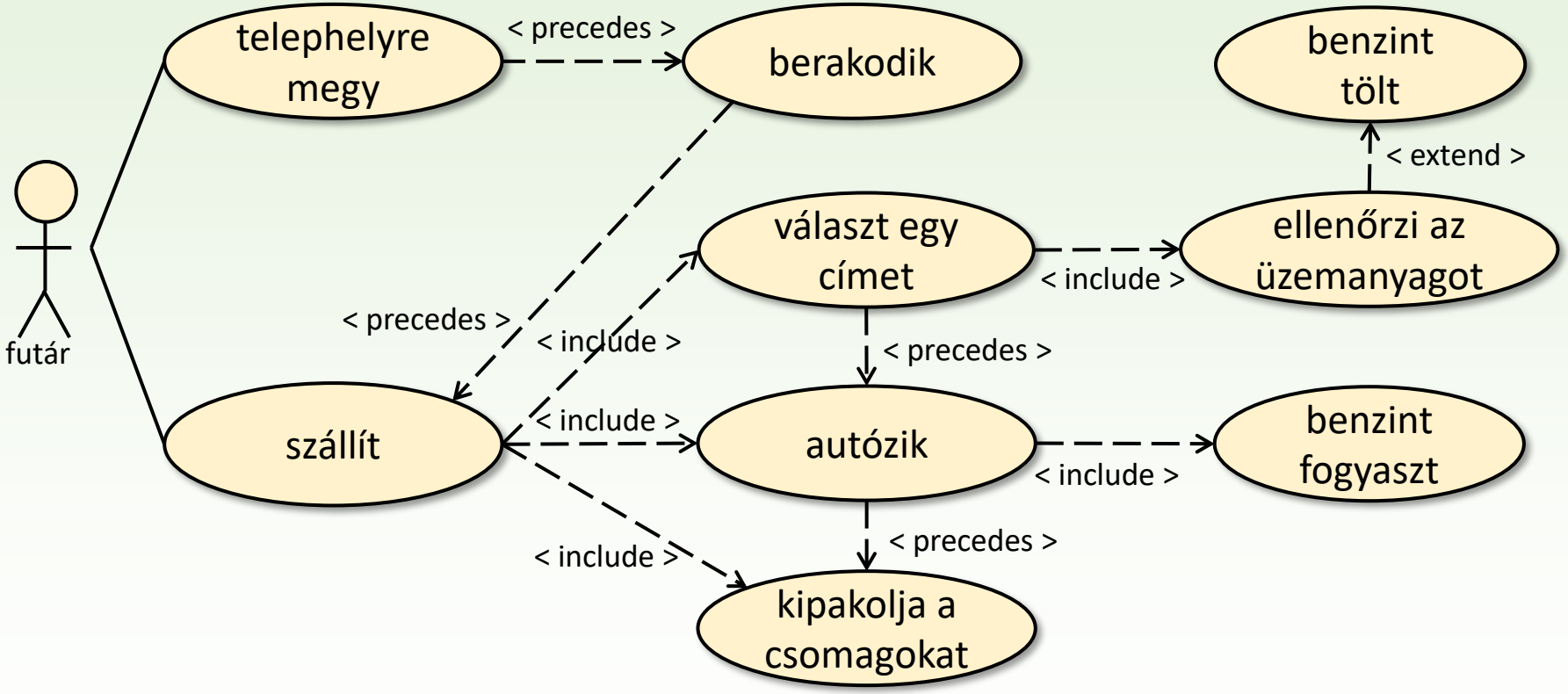
Csomagszállítás

Egy csomag kiszállító futár egy telephelyről hordja szét a megrendelt csomagokat különböző benzinkutakhoz telepített PickPack pontokra. (Tehát minden kiszállítási címen tankolni is lehet). A csomagokra ráírták a kiszállítás címét, és ebből kiszámolható, hogy mekkora távolságot kell autóznia a futárnak az aktuális tartózkodási helyétől a kiválasztott címig (km-ben).

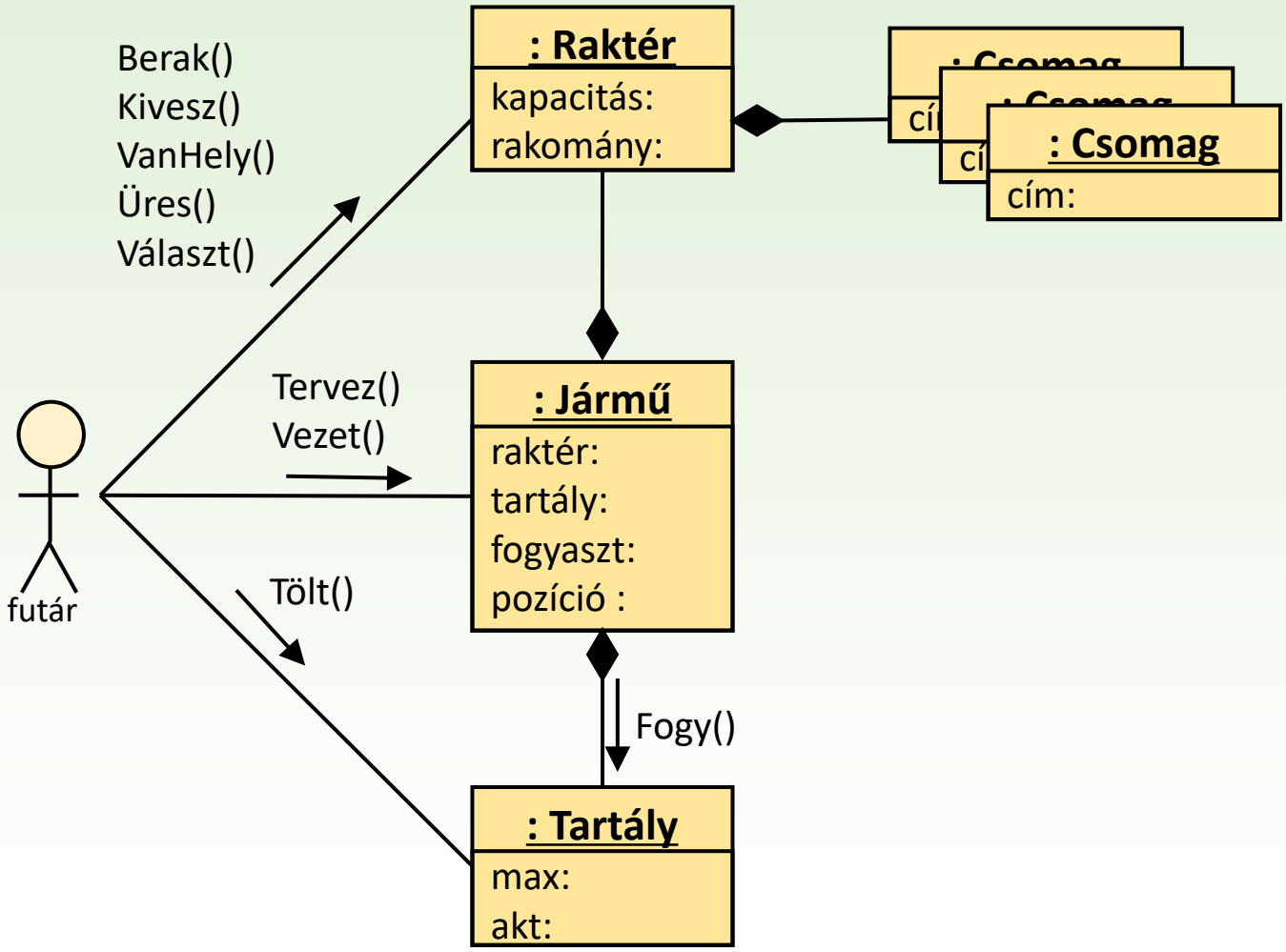
A futár bepakol annyi csomagot a járművé rakterébe, amennyit csak tud, majd a következők szerint jár el: kiválasztja a címét az egyik csomagjának (ha üres a rakodótér, akkor a telephelyének); ellenőrzi a benzinszintet, hogy elegendő-e a kiszállításhoz (ha nem, akkor tankol); elautózik a kiválasztott címre (ennek következményeképp csökken a benzinszint); majd kipakolja az adott címre küldött csomagokat.

A járműnek van rakodótere és egy benzintartálya. A rakodótér megadott számú csomagot képes tárolni. A tartályba a maximális benzinszint figyelembe vételével tankolhatunk. Ismert a jármű fogyasztása (liter/km mértékegységben).

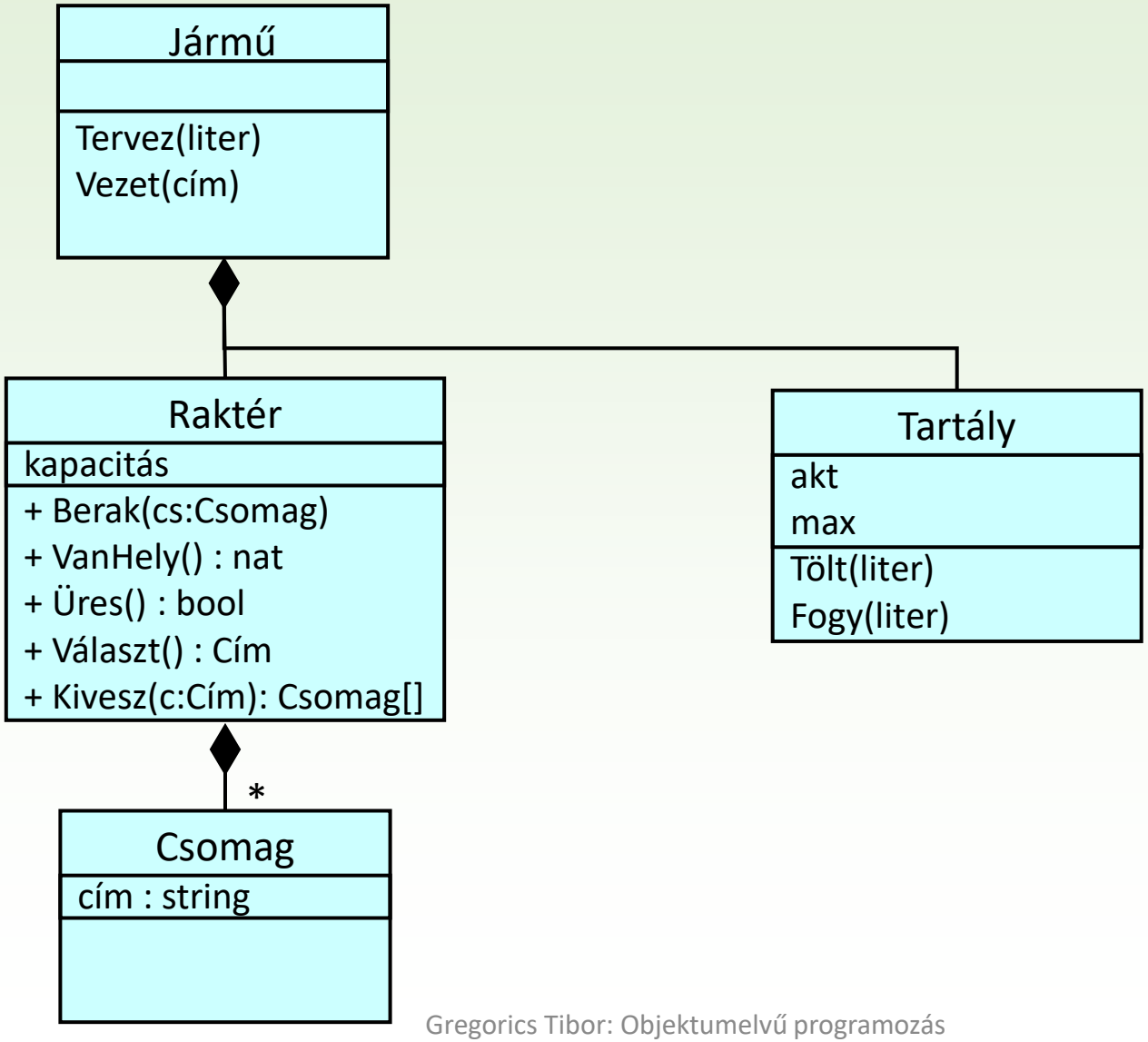
Csomagszállítás

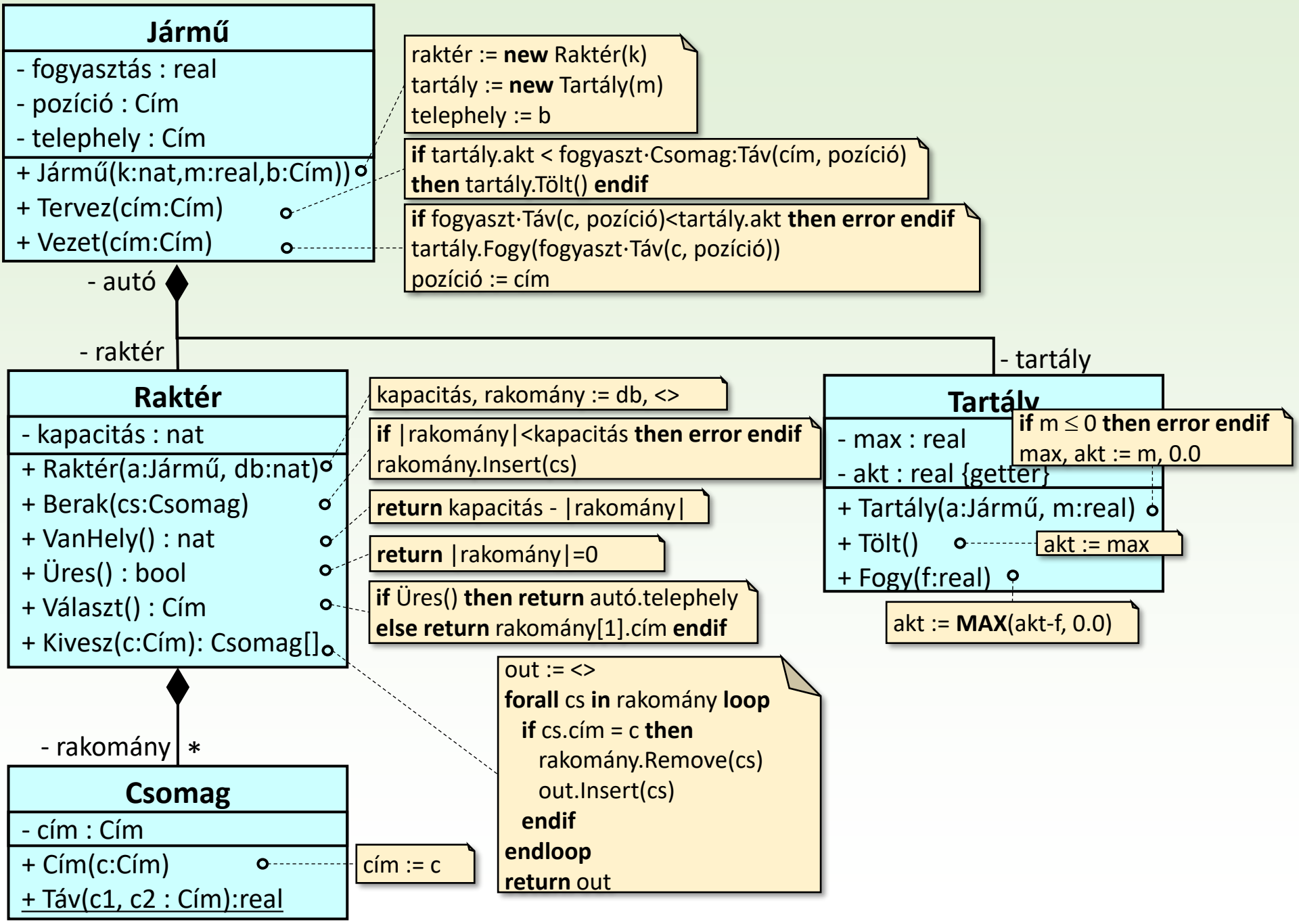


Csomagszállítás



Csomagszállítás




Jármű
- fogyasztás : real
- pozíció : Cím
- telephely : Cím
+ Jármű(k:nat,m:real,b:Cím))
+ Tervez(cím:Cím)
+ Vezet(cím:Cím)
raktér := new Raktér(k)
tartály := new Tartály(m)
telephely := b
if tartály.akt < fogyaszt·Csomag:Táv(cím, pozíció)
then tartály.Tölt() endif
if fogyaszt·Táv(c, pozíció)<tartály.akt then error endif
tartály.Fogy(fogyaszt·Táv(c, pozíció))
pozíció := cím
- autó
- raktér
- tartály
Raktér
- kapacitás : nat
+ Raktér(a:Jármű, db:nat)
+ Berak(cs:Csomag)
+ VanHely() : nat
+ Üres() : bool
+ Választ() : Cím
+ Kivesz(c:Cím): Csomag[]
kapacitás, rakomány := db, <>
if |rakomány|<kapacitás then error endif
rakomány.Insert(cs)
return kapacitás - |rakomány|
return |rakomány|=0
if Üres() then return autó.telephely
else return rakomány[1].cím endif
out := <>
forall cs in rakomány loop
if cs.cím = c then
rakomány.Remove(cs)
out.Insert(cs)
endif
endloop
return out
Tartály
- max : real
- akt : real {getter}
+ Tartály(a:Jármű, m:real)
+ Tölt()
+ Fogy(f:real)
if m ≤ 0 then error endif
max, akt := m, 0.0
akt := max
akt := MAX(akt-f, 0.0)
- rakomány
*
Csomag
- cím : Cím
+ Cím(c:Cím)
+ Táv(c1, c2 : Cím):real
cím := c