Szoftvertechnológia és -technikák

2. Előadás

Bevezetés az UML világába, osztálydiagram

BME AUT
Automatizálási és
Alkalmazott
Informatikai Tanszék

Tartalom

Modellezés - bevezetés

UML

Osztálydiagram

Modellezés

Szoftverfejlesztés

- Készítsünk egy programot, ami kiszámolja és kiírja egy adott szám faktoriálisát!

```
class Faktorialis
{
    public static void main(String args[])
    {
        int i;
        int fact = 1;

        int number = 5;
        for (i = 1; i <= number; i++)
        {
            fact = fact * i;
        }
        System.out.println("A(z) " + number + " faktoriálisa: " + fact);
    }
}
```

Szoftverfejlesztés

- Készítsünk programot, ami kiszámolja a binomiális tételt bekért paraméterekkel!
  $\sum_{k=0}^{n}\binom{n}{k}x^k a^{n-k}$

- Készítsünk programot, ami kiszámol egy tetszőleges matematikai műveletsort!
  (Az elérhető függvények listája előre adott)

- Készítsünk egy tőzsdei trendeket előrejelző, pénzügyi tanácsadó alkalmazást mobilra!

Szoftverfejlesztés

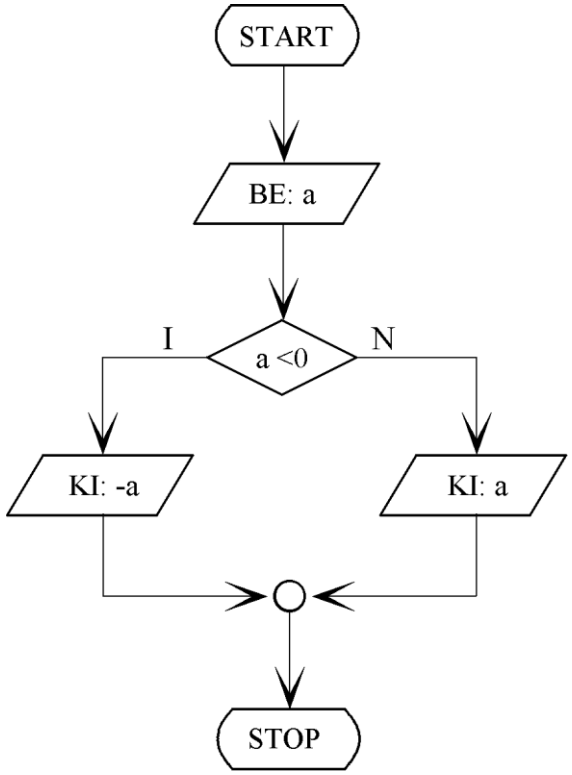
- Tervezés nélkül is lehet építeni házat
  - Fatárolóhoz elég lehet
  - Ha egyedül építi az ember
  - Azonnal összeomlik…
  - … vagy csak később?
- Több szintes családi ház
  - Többen építik
    - Sorrend: A tetőfedő csak az elején ér rá?
    - Értelmezés: Tűzfalra néző ablak
  - Élni is szeretnénk benne
  - Sok évig

Szoftverfejlesztés nagyban

- Szoftver
  - Kis feladat → “Csak meg kell írni”
  - Komplex feladat → Érdemes megtervezni
- Tervezés
  - Ha a kódméret/feladat komplex
  - Ha egyeztetni kell a megrendelővel
  - Ha meg kell becsülni a költségeket
  - Ha több ember fejleszti, új belépők is lehetnek
  - Ha segédmunkások is vannak
- Megoldás
  - Modellezés

Modellezés

- Modell: a valóság egy egyszerűsített nézete
  - Absztrakciós szint növekedés
  - Hangsúly azon, ami fontos…
  - … elrejtve azt, ami nem
- Modell a szoftverfejlesztéshez
  - A struktúrához?
  - A folyamatokhoz?
  - A viselkedéshez?
  - … Vagy mindenhez ezek közül?



Unified Modeling Language

Bevezetés

Szoftvermodellezés

- Igény a szoftverek tervezésére
  - Magas absztrakciós szint, vizuális ábrázolás
  - ‘90-es évektől több modellező nyelv: kaotikus helyzet
  - 1997. – Unified Modeling Language (UML)
- Unified Modeling Language
  - Cél: szoftverfejlesztés és dokumentáció modellezése
  - Általános modellezőnyelv
  - A szabvány modellezés terén
  - Jelenlegi verzió: 2.5.1.
    https://www.omg.org/spec/UML/2.5.1/PDF
  - Jó magyarázatok és példák:
    https://www.uml-diagrams.org
  - Modellezőeszköz a félév során:
    Modelio (https://www.modelio.org/)
    - Részletek gyakorlaton!



UML

- UML használatának előnyei
  - Vázlat
    - Implementáció előtt:
      - Probléma jobb megértése
      - Hibák/hiányosságok feltárása
    - Implementáció után
      - Dokumentálás
    - Kommunikáció
      - Megrendelővel
      - Fejlesztőkkel
  - Tervrajz
    - Terv alapján könnyebb dolgozni (architect vs. programozó)

UML diagrammok

- Struktúra
  - Osztálydiagram (Class diagram)
  - Komponensdiagram (Component diagram)
  - Telepítési diagram (Deployment diagram)
  - Objektumdiagram (Object diagram)
  - Csomagdiagram (Package diagram)
  - Összetett struktúradiagram (Composite Structure diagram)
  - Profildiagram (Profile diagram)

UML diagramok II.

- Viselkedés
  - Aktivitásdiagram (Activity diagram)
  - Használati eset diagram (Use case diagram)
  - Állapotgép (State machine)
  - Interaktivitás
    - Szekvenciadiagram (Sequence diagram)
    - Kommunikációs diagram (Communication diagram)
    - Interakció áttekintő diagram (Interaction overview diagram)
    - Időzítő diagram (Timing diagram)

UML

- Mit ad az UML?
  - Jelölésrendszer, közös nyelv
  - Magas absztrakciós szint, tömör leírás
  - Szabványos keret
  - Segít szemléltetni a szoftvertervezési módszereket
- Mit nem ad az UML?
  - Nem oldja meg a problémát, csak segít leírni azt
  - Nem mutatja meg, *hogyan* tervezzünk jól
    - Ezért tanulunk a Clean code-ról, a SOLID elvekről,…
    - Ezért kellenek a tervezési minták

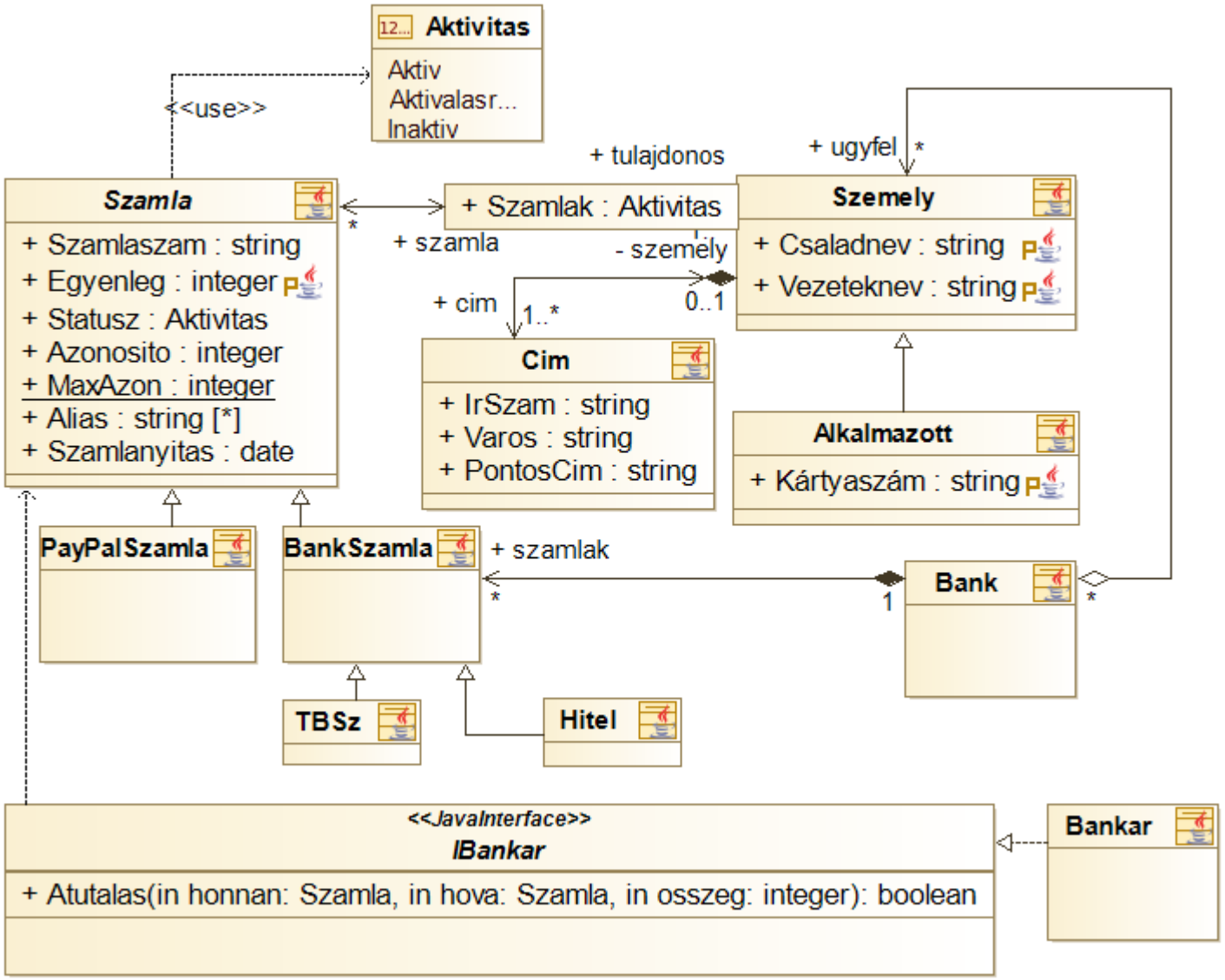
Osztálydiagram

Class diagram

Osztálydiagram

- A leggyakrabban használt, legfontosabb UML diagram
- OOP-alapon készült rendszerekhez
- Struktúra megadására (osztályok, tagváltozók, metódusok, kapcsolatok)
- Progamozási nyelvtől független!
  - Generálható belőle forráskód (metódusoknál csak a váz)

Osztálydiagram: Banki adminisztrációs rendszer



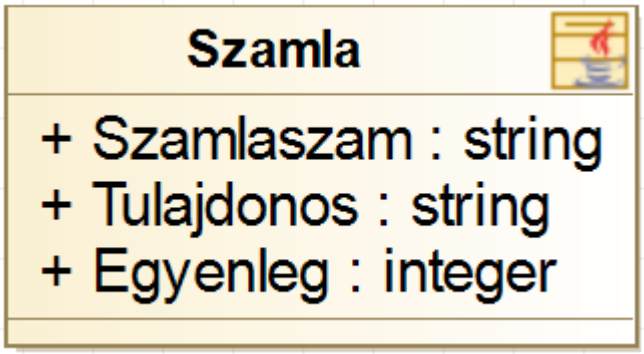
Modellelemek

Osztályok

Számlaszám	Tulajdonos neve	Egyenleg (Ft)
11742232-15393276	Nagy Lajos	12 000 343
11742232-75343245	Hunyadi Mátyás	4 556
11742232-35393127	Károly Róbert	443 227 786

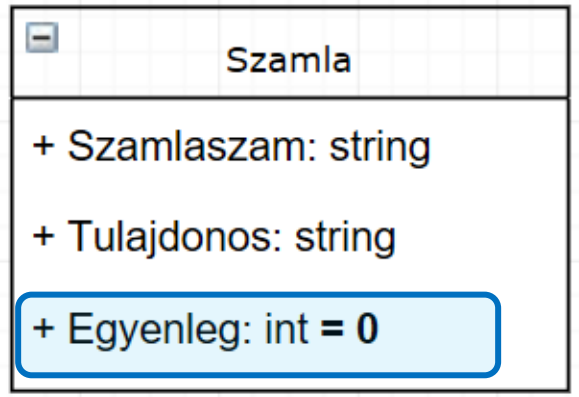
- Készítsünk hozzá egy osztályt és tagváltozókat!
  - > Osztály (class) és tagváltozó (attribute)

```
1  public class Szamla {
2
3      public String Szamlaszam;
4      public String Tulajdonos;
5      public int Egyenleg;
6
7  }
```



Kezdőérték

- Legyen az egyenleg alapesetben 0 Ft!
  - *Sajnos a Modelio eszköz nem jeleníti meg az ábrán!*

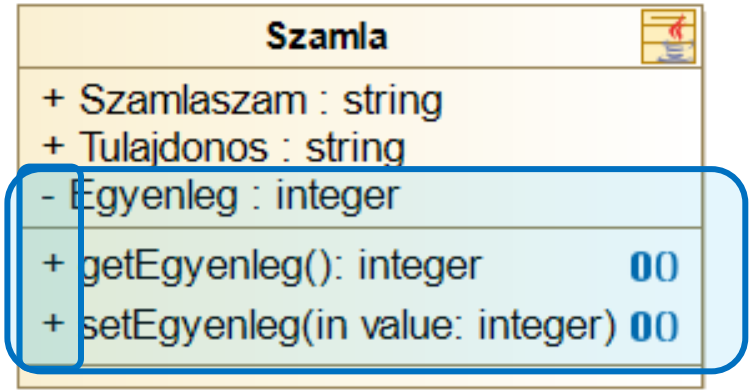


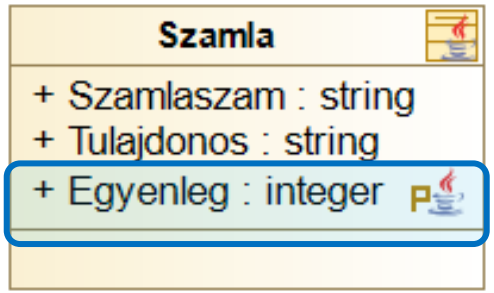
  - A generált kód:

```
1  import com.modeliosoft.modelio.javadesigner.annota
2
3  @objid ("84fe0cc8-8a45-4874-9b29-6c491370c38b")
4  public class Szamla {
5      @objid ("e2c21051-b938-40b4-b271-48f932146246")
6      public String Szamlaszam;
7
8      @objid ("f15659a4-cc45-45be-bc79-597568508df9")
9      public String Tulajdonos;
10
11     @objid ("d662343e-735f-44fa-8c68-1db19fdf66c2")
12     public int Egyenleg = 0;
13
14 }
15
```

Láthatóság

- Az egyenleg megváltoztatását és lekérdezését válasszuk szét!
  - Metódus (method)
  - Láthatóság (visibility/ member access modifier)
  - Privát tagváltozó + publikus metódusok
  - Tagváltozó → property



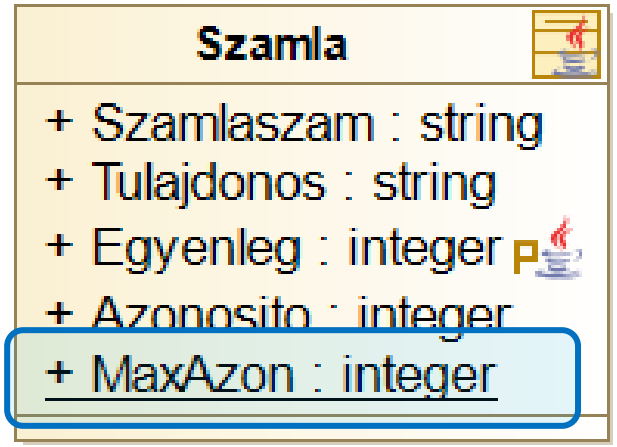



```
12      private int Egyenleg = 0;
13
14      public int getEgyenleg() {
15          return this.Egyenleg;
16      }
17      public void setEgyenleg(int value) {
18          this.Egyenleg = value;
19      }
```

Statikus tagváltozók

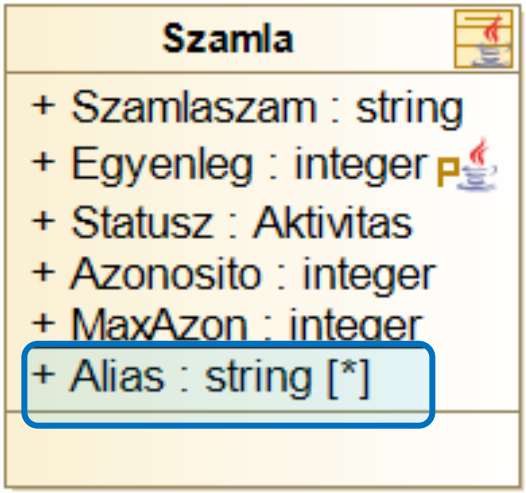
- Minden számlának legyen egy egyedi, belső azonosítója. Számla létrehozásakor az eddigi maximum azonosítóból számoljuk az új számla azonosítóját!
  - A számlaazonosító példány szintű adat (minden számlának más)
  - A max számlaazonosító viszont *osztály* szintű!
  - Statikus tagváltozó kell (aláhúzás!)
  - A számolási logika nem a modell része!

```
public class Szamla {
    ...

    public int Azonosito;
    public static int MaxAzon;

    ...
}
```



Multiplicitás

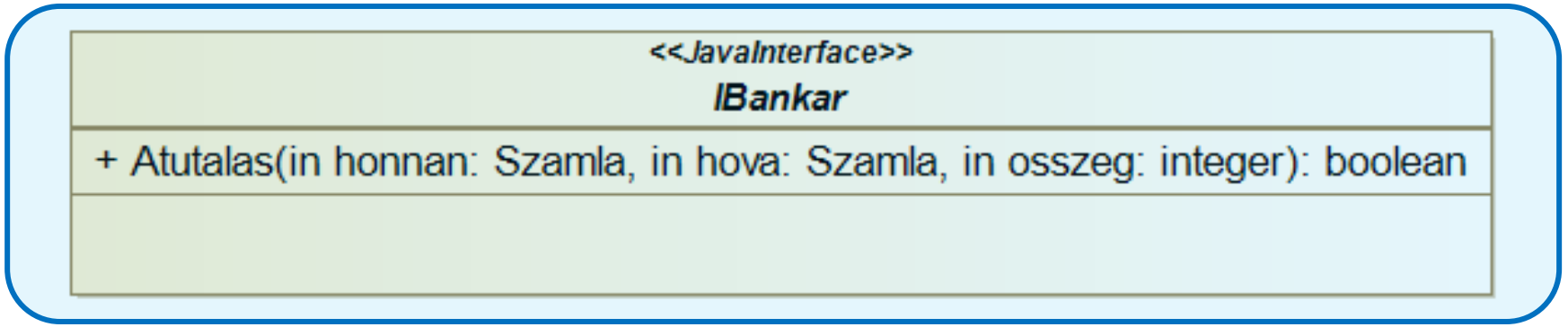
- Lehessen megadni a számlához több alias nevet!
  - Multiplicitás (multiplicity)
  - Attribútum többes multiplicitással
    - 0..1
    - 1 ..1 (alapértelmezett)
    - 0 ..* (lista generálódik belőle!)
    - n .. m



Interfész

- Készítsünk egy interfészt, ami támogatja a számlák közti átutalás műveletét!
  - > Művelet (operation)
    - Művelet - definíció vs. Metódus - implementáció
  - > A paraméter típusa nem beépített típus!
  - > Szignatúra:

```
+ Atutal( honnan: Szamla,
           hova: Szamla,
           osszeg: integer): bool
```



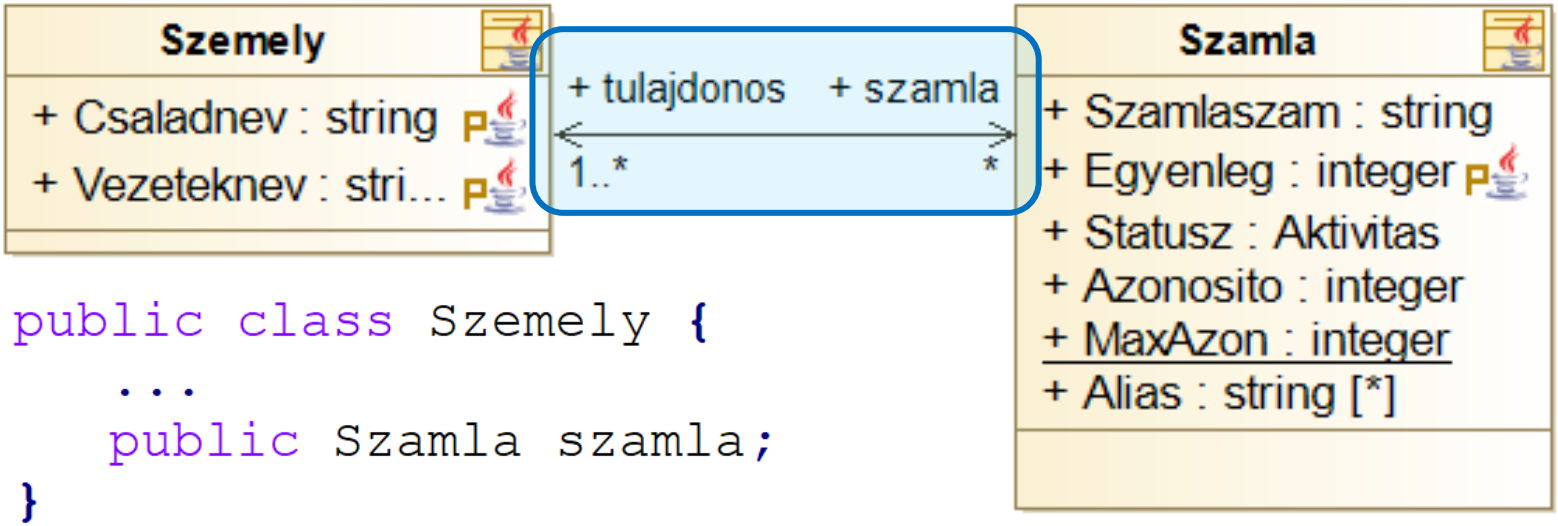
Osztályok és interfészek

- Osztályok és interfészek
  - Fejléc
  - Tagváltozók
    - `[Láthatóság] [Név]: [Típus][Multiplicitás] [= kezdőérték]`
    - Statikus: aláhúzás
  - Metódusok/Műveletek
    - `[Láthatóság] [Név]([paraméterek])[: visszatérési érték]`
  - Láthatóság
    - Public: +
    - Private: -
    - Protected: #
    - Package: ~

Kapcsolatok

Asszociáció

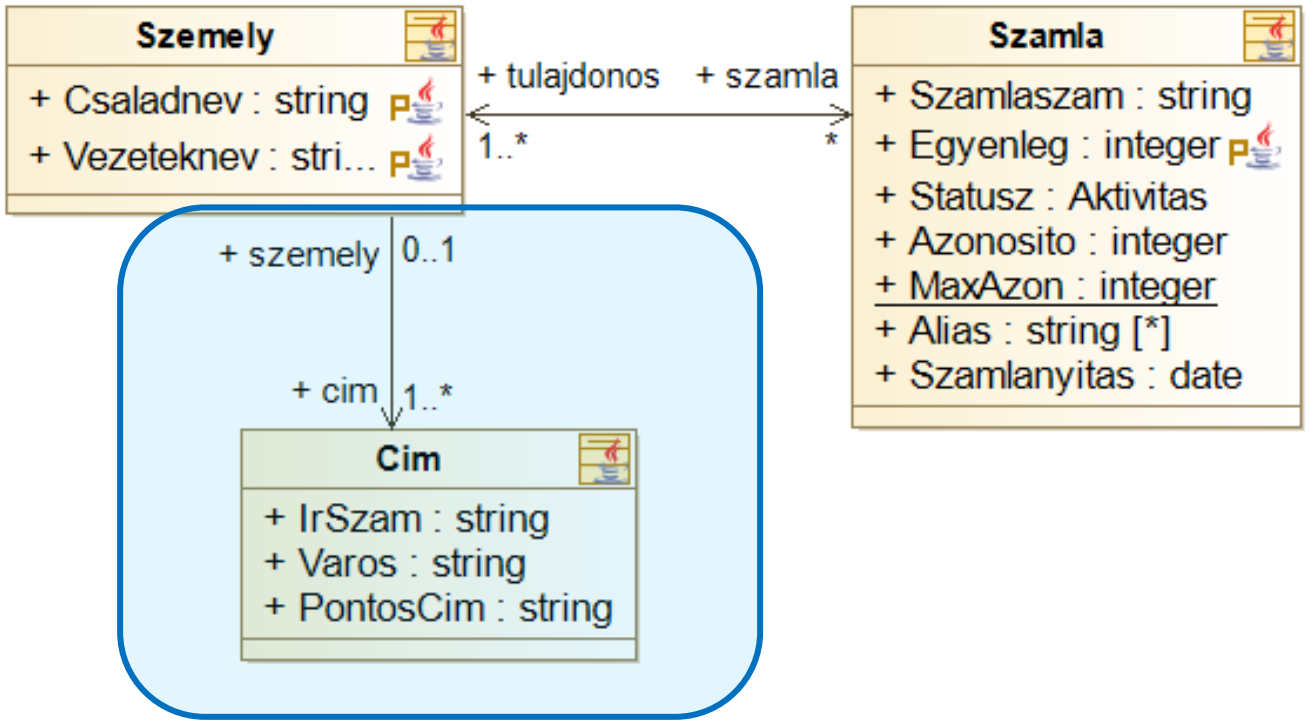
- Támogassuk, hogy egy személynek több számlája is lehessen!
  - > Asszociáció (association)
  - > Ki kell emelni a személyt az osztályból!
  - > Szerepnév (role name) és multiplicitás (multiplicity)
  - > Kódban: tagváltozó




```
public class Szemely {
    ...
    public Szamla szamla;
}
```

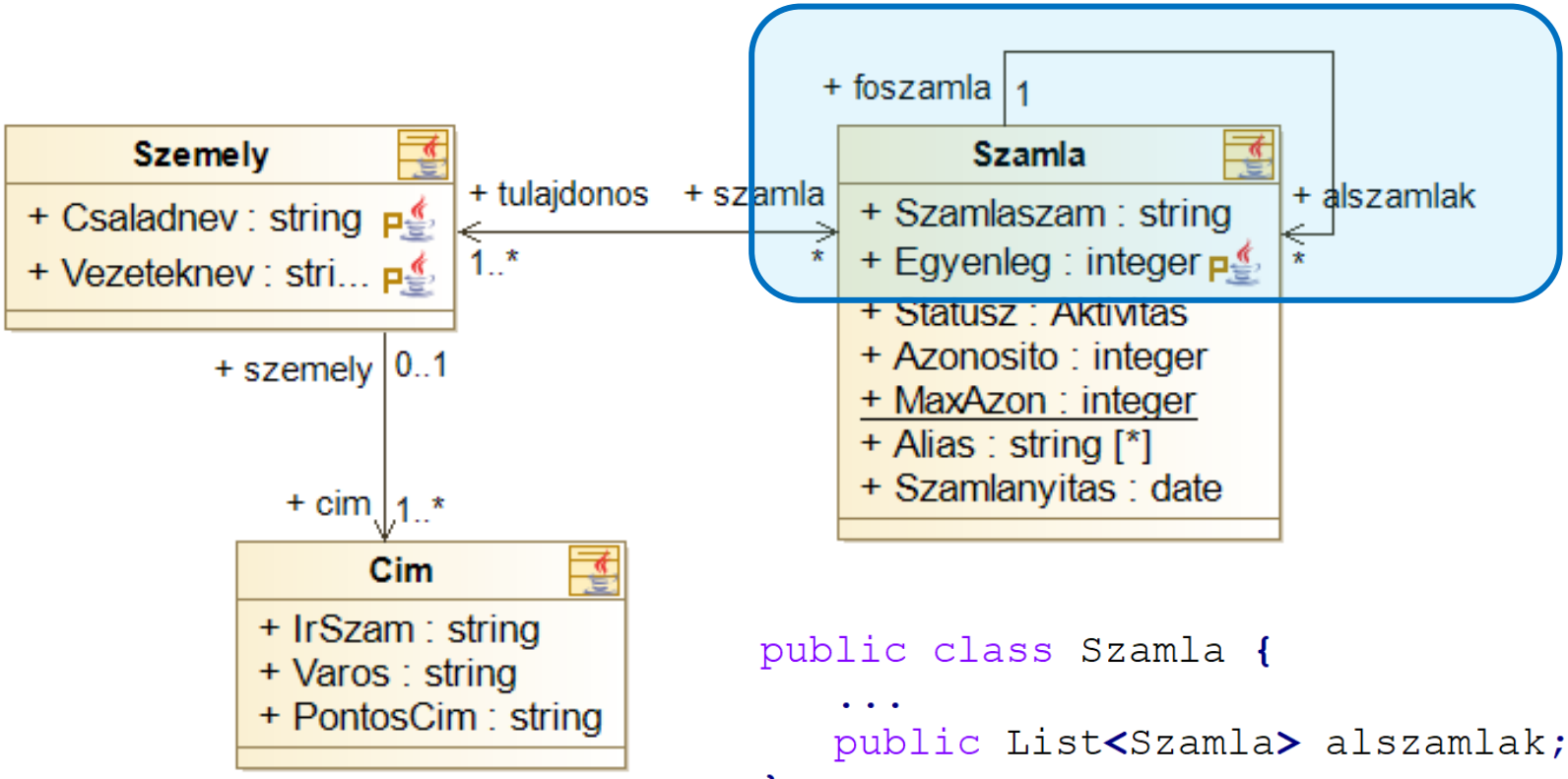
Asszociáció

- Lehessen megadni az egyes emberek lakcímét!
  - > Egy irányban navigálható kapcsolat



Asszociáció

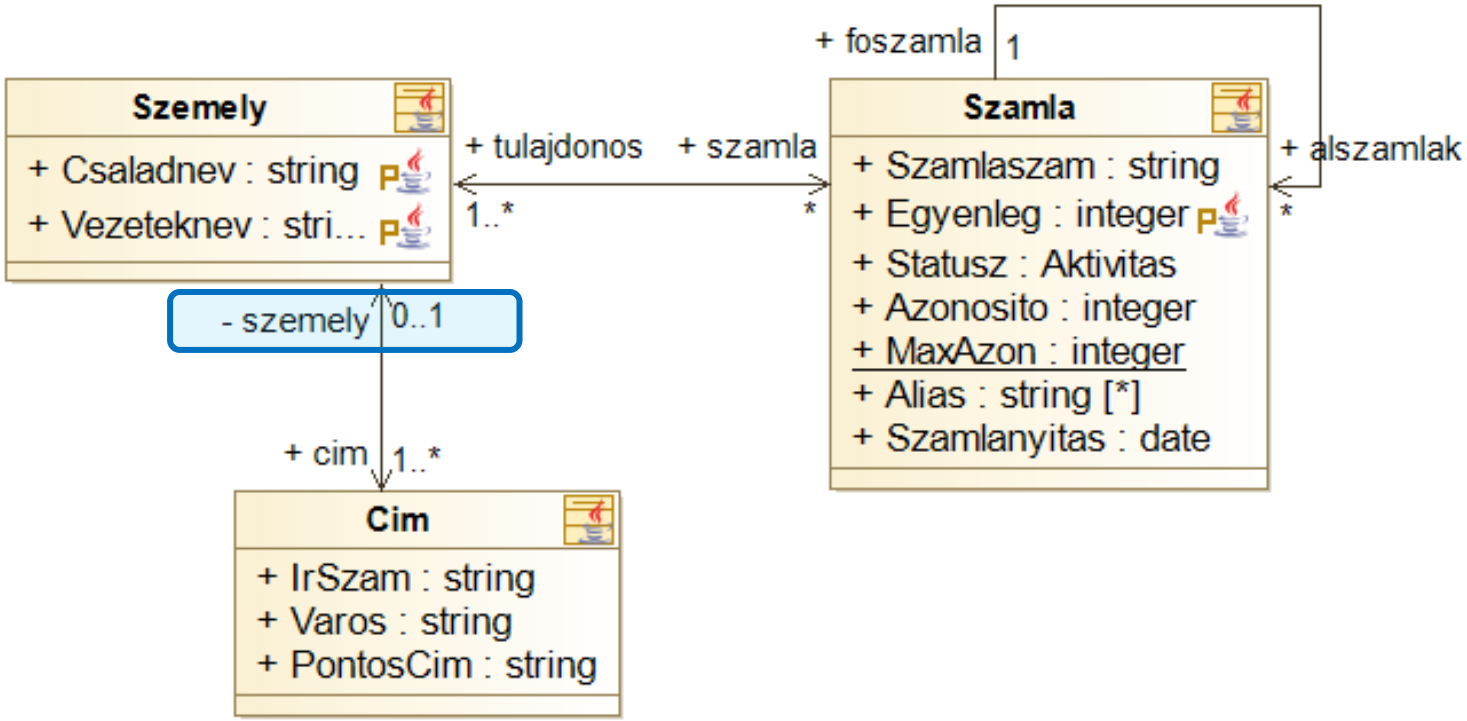
- Lehessen megadni alszámlákat!
  - Hurokél




```
public class Szamla {
    ...
    public List<Szamla> alszamlak;
}
```

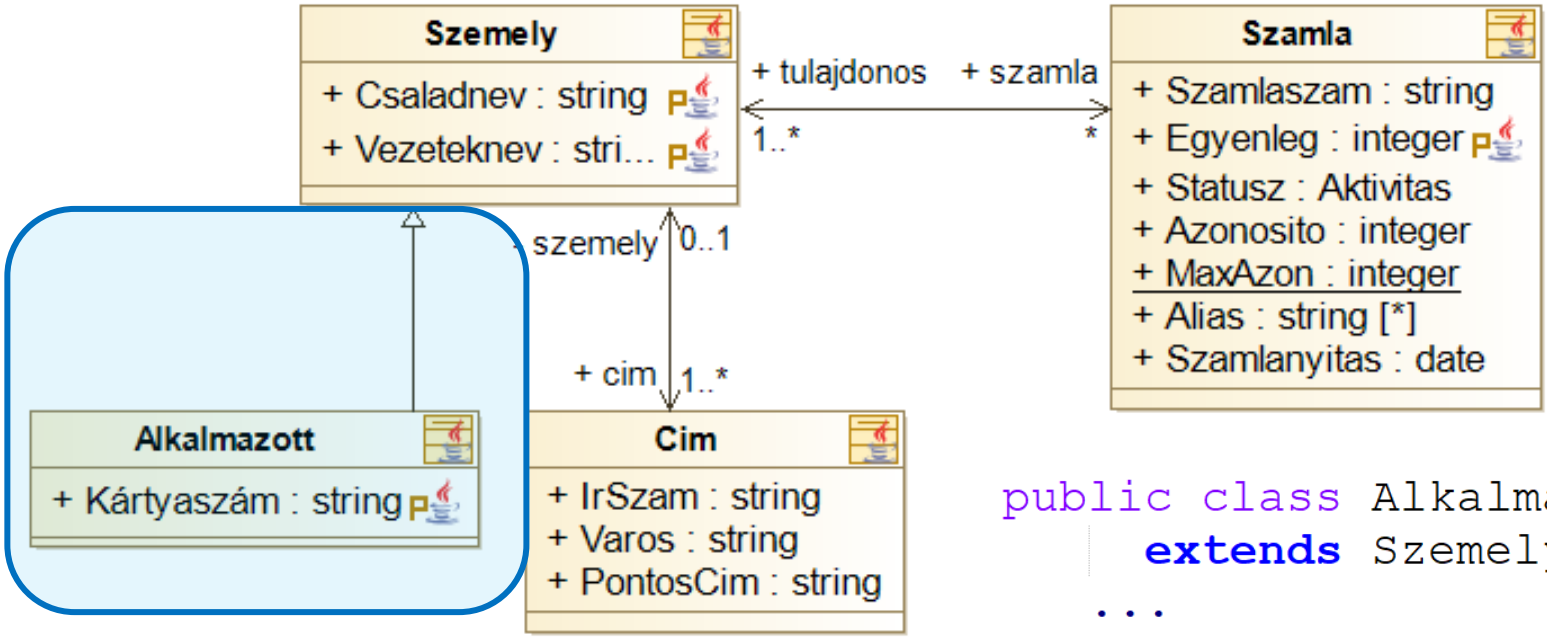
Asszociáció

- A címtől lehessen lekérdezni, melyik személyhez tartozik, de mindezt csak a cím entitáson belülről!
  - Láthatóság a navigáción!



Általánosítás/öröklés

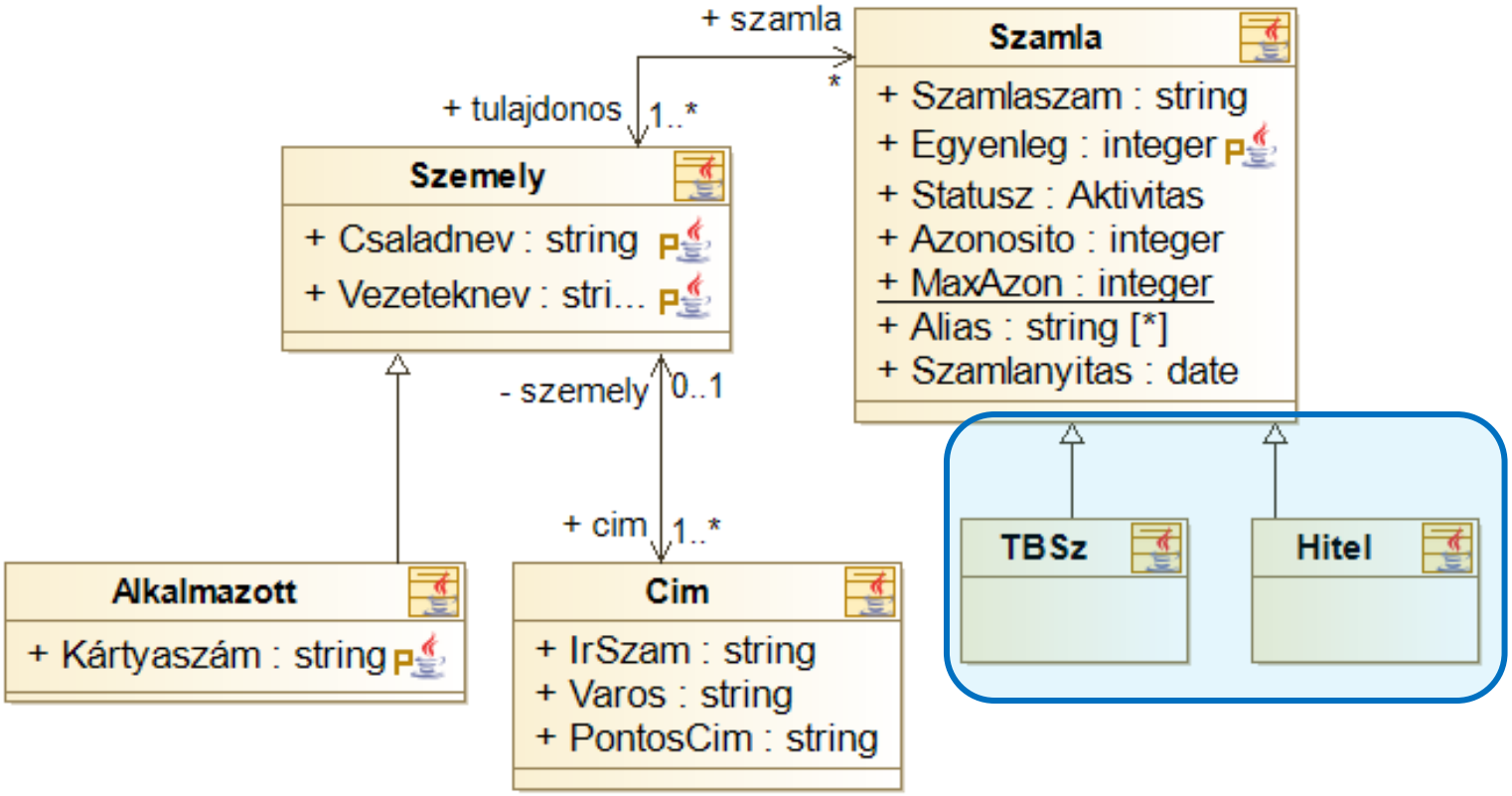
- Tartsuk nyilván a banki dolgozókat is! Nekik is lehet számlájuk, címük, de van azonosítókártyájuk is!
  - > Általánosítás/öröklés (generalization/inheritance)
  - > Kódban: öröklés a két osztály közt




```
public class Alkalmazott
      extends Szemely {
    ...
}
```

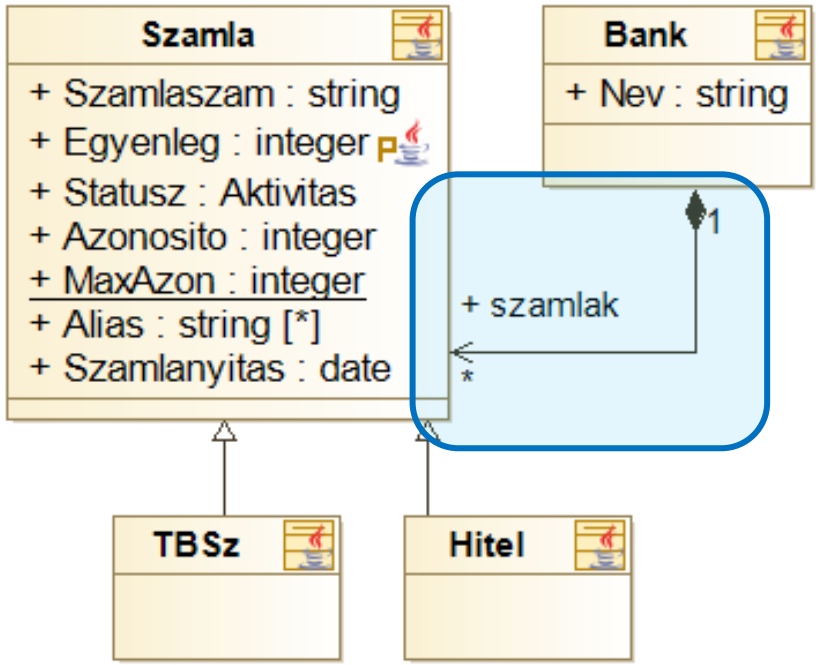
Általánosítás/öröklés

- Legyenek külön számlatípusaink! (TBSz, Hitel)
  - > Általánosítás/öröklés?



Tartalmazás

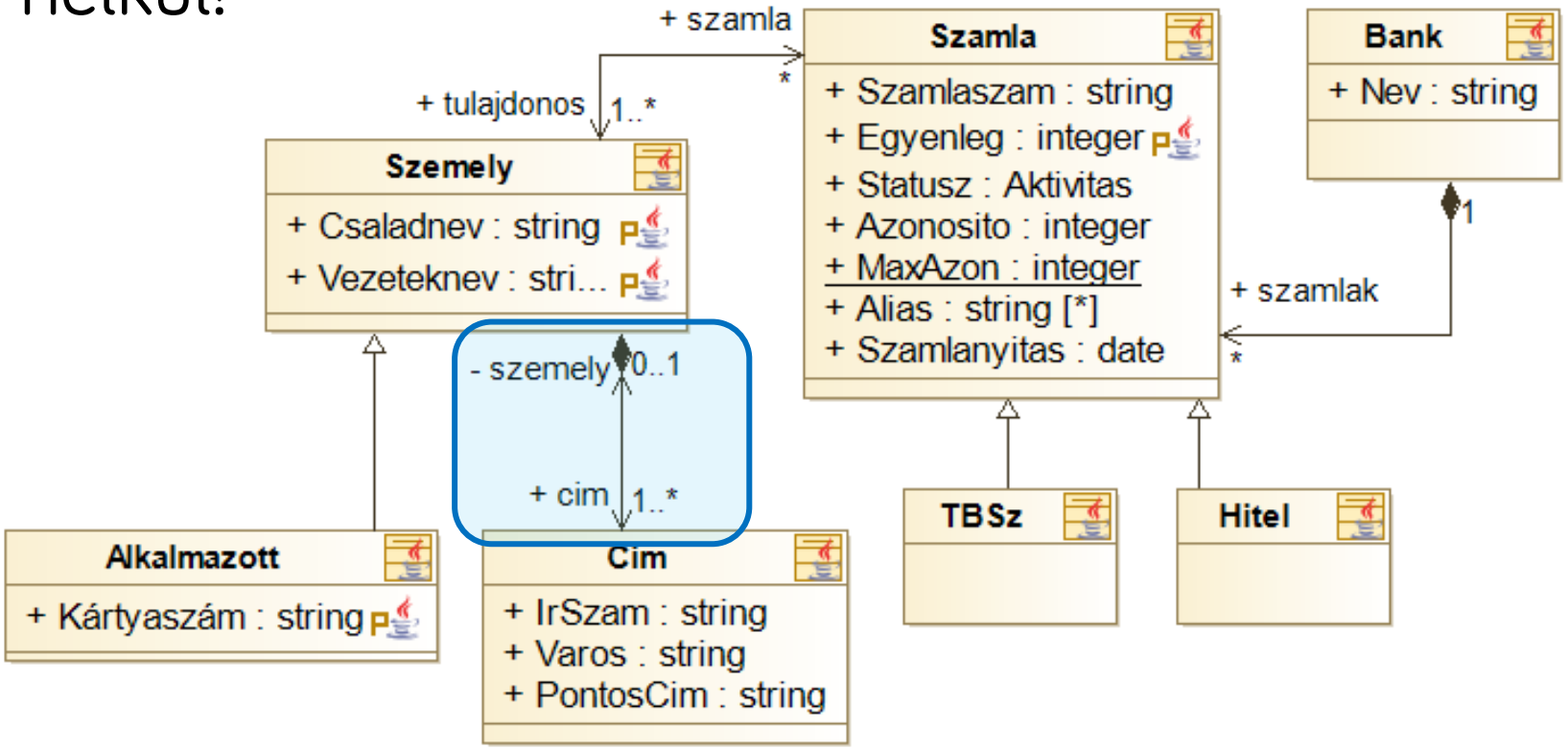
- Minden számla tartozzon egy bankhoz!
  - Egy számla nem tartozhat két bankhoz
  - Ha a bankot töröljük, a számlát is törölni kell!
  - Kompozíció (composition)
  - Kódban: tagváltozó




```
public class Bank {
    ...
    public List<Szamla> szamlak;
}
```

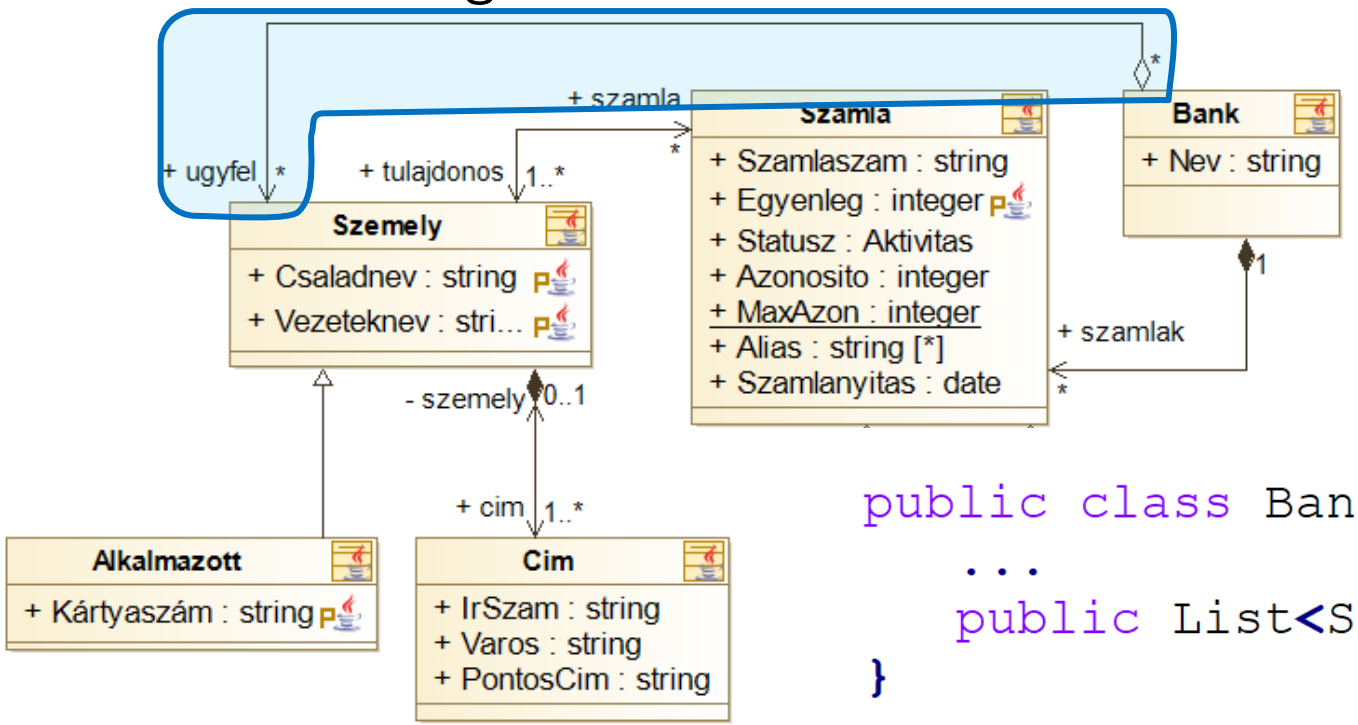
Tartalmazás

- Egy cím csak egy személyhez tartozhasson és ne tároljunk feleslegesen címeket, személy nélkül!



Tartalmazás

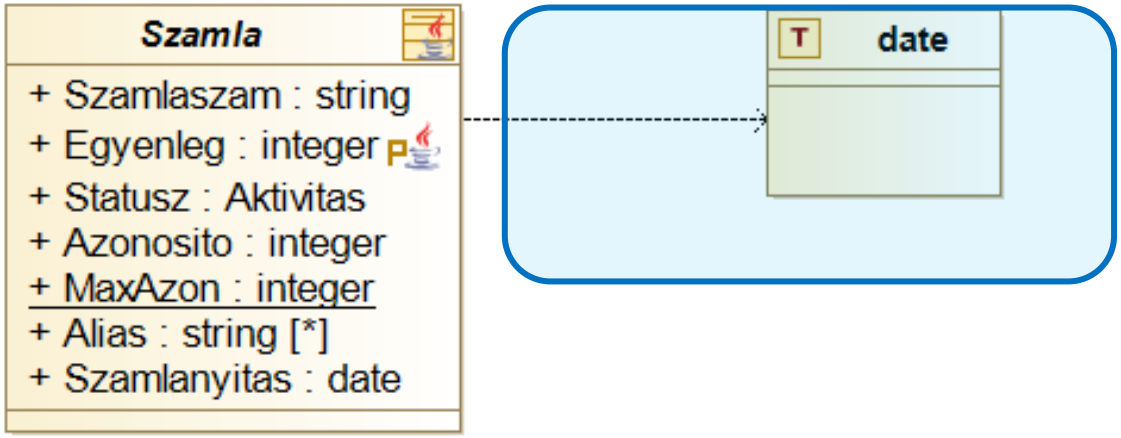
- A bank tartsa nyilván az ügyfeleit, akkor is, ha azok megszűntetik a számlájukat! Egy ügyfél több bank ügyfele is lehet.
  - Megosztott tartalmazás (aggregation)
  - Kódban: tagváltozó




```
public class Bank {
    ...
    public List<Szemely> ugyfel;
}
```

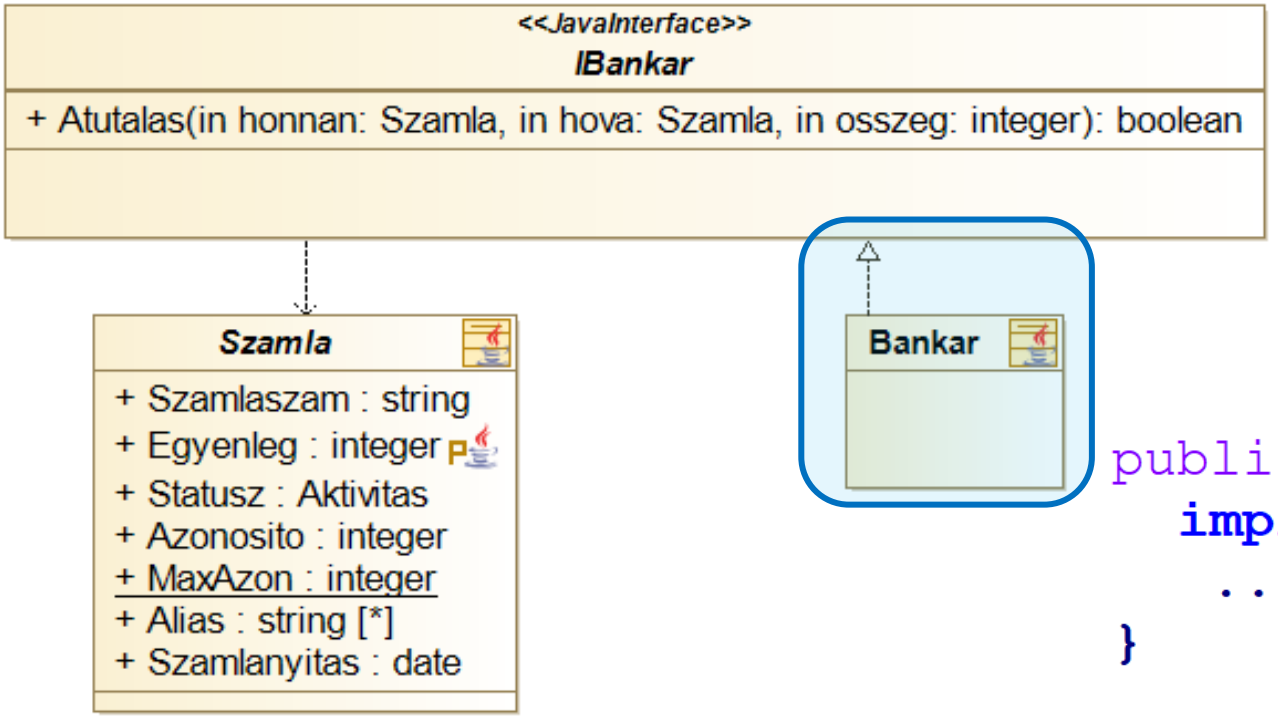
Függőség

- Lehessen megadni a számlanyitás dátumát!
  - A típus itt egy saját típus (modellelem)
  - Függőség (dependency) kapcsolat
  - Jellemzően nem szoros, hanem ideiglenes kapcsolat
  - A függőből mutat afelé, amitől függünk
  - Kódban: import/using



Interfészek II.

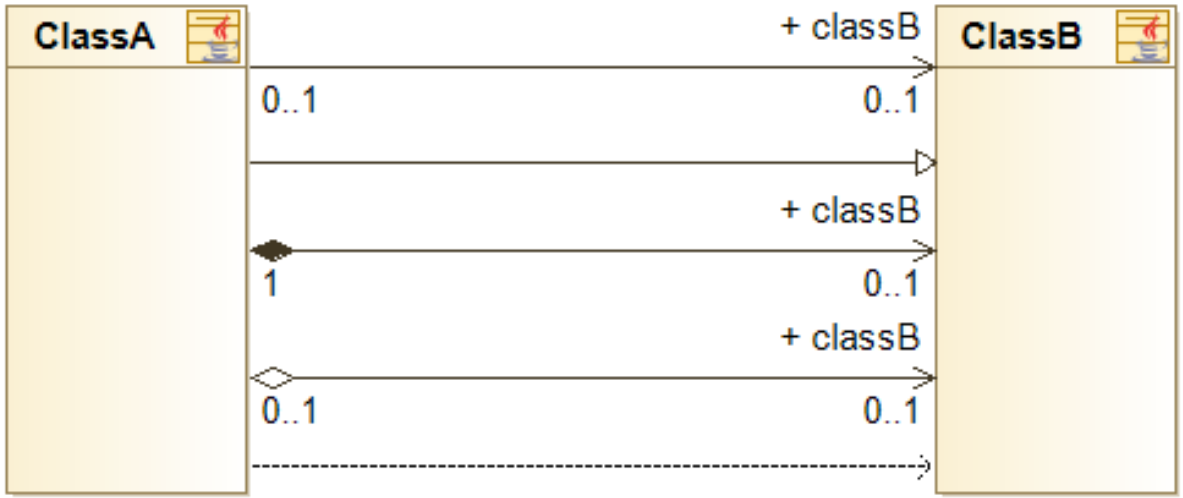
- Térjünk vissza az IBankar interfészhez:
  - Jelöljük a kapcsolatot a számlával!
  - Valósítsuk meg egy osztályban az interfészt!
  - Interfész megvalósítás (interface realization)
  - Kódban: implements/”:”




```
public class Bankar
    implements IBankar {
      ...
}
```

Kapcsolatok

- Kapcsolat típusok
  - Asszociáció
  - Általánosítás/ Öröklés
  - Kompozíció
  - Aggregáció
  - Függőség
  - Interfész megvalósítás



- Multiplicitás
- Navigálás
  - Szerepnév
  - Navigálhatóság

A kapcsolatok szemantikája

- „Is-a” kapcsolat
  - Általánosítás/interfész megvalósítás (generalization/interface realization)
  - Közös funkcionalitás
  - Ősosztállyal felcserélhetőség (Liskov)
  - Kibővített működés
- „Has-a” kapcsolat
  - Kizárólagos: kompozíció (containment)
    - Tartalmazó törlése törli a tartalmazottat
    - Tartalmazó egyedi
  - Megosztott: aggregáció (aggregation)
- Nem tartalmazás, nem öröklés, de (részben) navigálható
  - Asszociáció (association)
  - Kódban hasonlít a tartalmazáshoz, szemantikai különbség
- Laza/ideiglenes kapcsolat
  - Függőség (dependency)

Köszönöm a figyelmet!

BME AUT

Feladat: weboldal szerkesztő alkalmazás

- Az oldalakon űrlapok (form) lehetnek (bármennyi), a formokban pedig beviteli elemek (input), min. 1
- Ha egy formot törlünk az oldalról, törlődik az összes benne lévő elem
- Beviteli elem a textbox, a dropdownlist és a button, a gombot meg lehet nyomni (push), a dropdownt le lehet gördíteni.
- A textbox pontosan 1, a dropdown min. 1, max. 10 szöveget tartalmaz
- Maga a weboldal, ill. az egyes beviteli elemek kikapcsolhatóak (enable/disable), de a formok nem.
- A textboxok közt csoportok definiálhatóak. A csoport max 5 elemből áll. Ha az egyik textbox ki van töltve, a többi automatikusan kapcsoljon ki! Egy textbox több csoportba is tartozhat
- Minden elemhez megadható egy stílus (css class). A stílusnak van neve és lehet egy vagy több stílus, aminek az értékeiből öröklődik (nem osztálydiagram szintű öröklés!)