

**Eötvös Loránd Tudományegyetem  
Informatikai Kar**

## **Programozási technológia II**

---

### **3. előadás**

## **Objektumorientált tervezés**

---

**© 2016 Giachetta Roberto**  
**groberto@inf.elte.hu**  
**<http://people.inf.elte.hu/groberto>**



# Objektumorientált tervezés

## Objektumok, osztályok

---

- Az objektumorientált tervezés során a rendszert az objektumok mentén építjük fel, ahol az objektum
  - a valóság absztrakcióját adja
  - biztosít egy elvárt funkcionalitást
  - adat és működés egymásba burkolásából épül fel
- Egy adott feladatban az objektumokat (osztályokat) be kell azonosítanunk azáltal, hogy
  - milyen funkciókat azonosítottunk az elemzés során, és azok milyen adatokkal dolgoznak
  - a valóságban milyen építőelemeket tudnánk megfeleltetni a funkcióknak

# Objektumorientált tervezés

## A tervezés fázisai

---

- A tervezés általában több fázisból épül fel, amely során finomítunk a terven
  - mivel már az első fázis alapján beazonosítani a szükséges objektumokat, és azok felépítését meglehetősen nehézkes
  - minden fázisban
    - bevezethetünk új osztályokat a beazonosított feladatokra
    - tovább pontosíthatjuk a már létező osztályok felépítését, az implementációs megkötéseket
    - felbonthatunk osztályokat, amennyiben túl bonyolulttá, túl szerteágazóvá válnak
    - összevonhatunk osztályokat, amennyiben túlzottan elaprózódnak

# Objektumorientált tervezés

## A tervezés alapelvei

---

- Az objektumorientált tervezés során öt alapelvet célszerű követnünk (*SOLID*):
  - *Single responsibility principle* (SRP): egy programegység csak egyvalamiért felelhet
  - *Open/closed principle* (OCP): a programegységek nyitottak a kiterjesztésre, de zártak a módosításra
  - *Liskov substitution principle* (LSP): az objektumok helyettesíthetők altípusaik példányával
  - *Interface segregation principle* (ISP): egy általános interfész helyett több kliens specifikus interfész
  - *Dependency inversion principle* (DIP): az absztrakciótól függünk, nem a konkretizációtól

# Objektumorientált tervezés

## Az architektúra

- *Szoftver architektúrának* nevezzük a szoftver fejlesztése során meghozott *elsődleges tervezési döntések* halmazát
  - azon döntések, amelyek megváltoztatása később jelentős újratervezését igényelné a szoftvernek
  - kihat a rendszer felépítésére, viselkedésére, kommunikációjára, nem funkcionális jellemzőire és megvalósítására
- A szoftver architektúra elsődleges feladata *a rendszer magas szintű felépítésének és működésének meghatározása*, a komponensek és relációk kiépítése
  - meghatározza a szolgáltatott és elvárt interfészek halmazát, a kommunikációs csatornákat és csatlakozási pontokat

# Objektumorientált tervezés

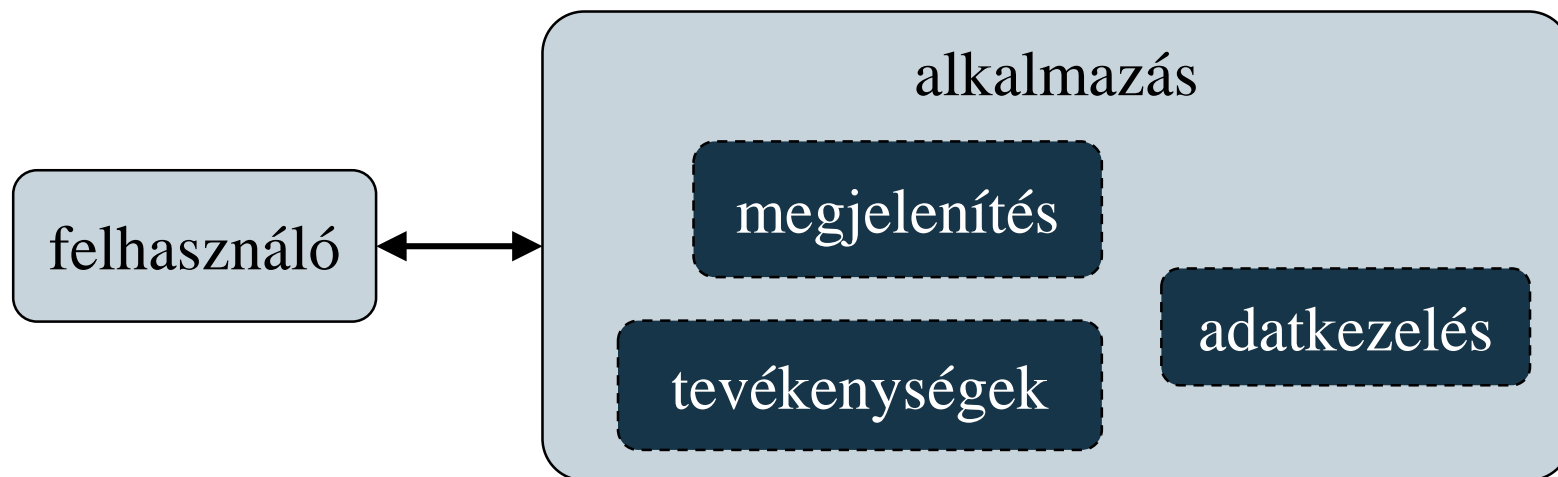
## Minták a tervezésben

- A szoftver architektúráját különböző szempontok szerint közelíthetjük meg, pl.:
  - a szoftver által nyújtott szolgáltatások (funkciók) szerint
  - a felhasználó és a futtató platform közötti tevékenységi szint szerint
  - az adatátadás, kommunikáció módja szerint
- Az architektúra létrehozása során mintákra hagyatkozunk, a szoftver teljes architektúráját definiáló mintákat nevezzük *architekturális mintáknak (architectural pattern)*, az architektúra alkalmazásának módját, az egyes komponensek összekapcsolását segítik elő a *tervminták (design pattern)*

# Objektumorientált tervezés

## A monolitikus architektúra

- Minden szoftver rendelkezik architektúrával
- A legegyszerűbb felépítést az *monolitikus architektúra* (*monolithic architecture*) adja
  - nem különböztetjük meg az egyes feladatköröket (pl. megjelenítés, adatkezelés), hanem egységesen kezeljük őket



# Objektumorientált tervezés

## 1. esettanulmány

*1. esettanulmány:* Készítsük el Marika néni kávézójának eladási nyilvántartását végigkövető programot.

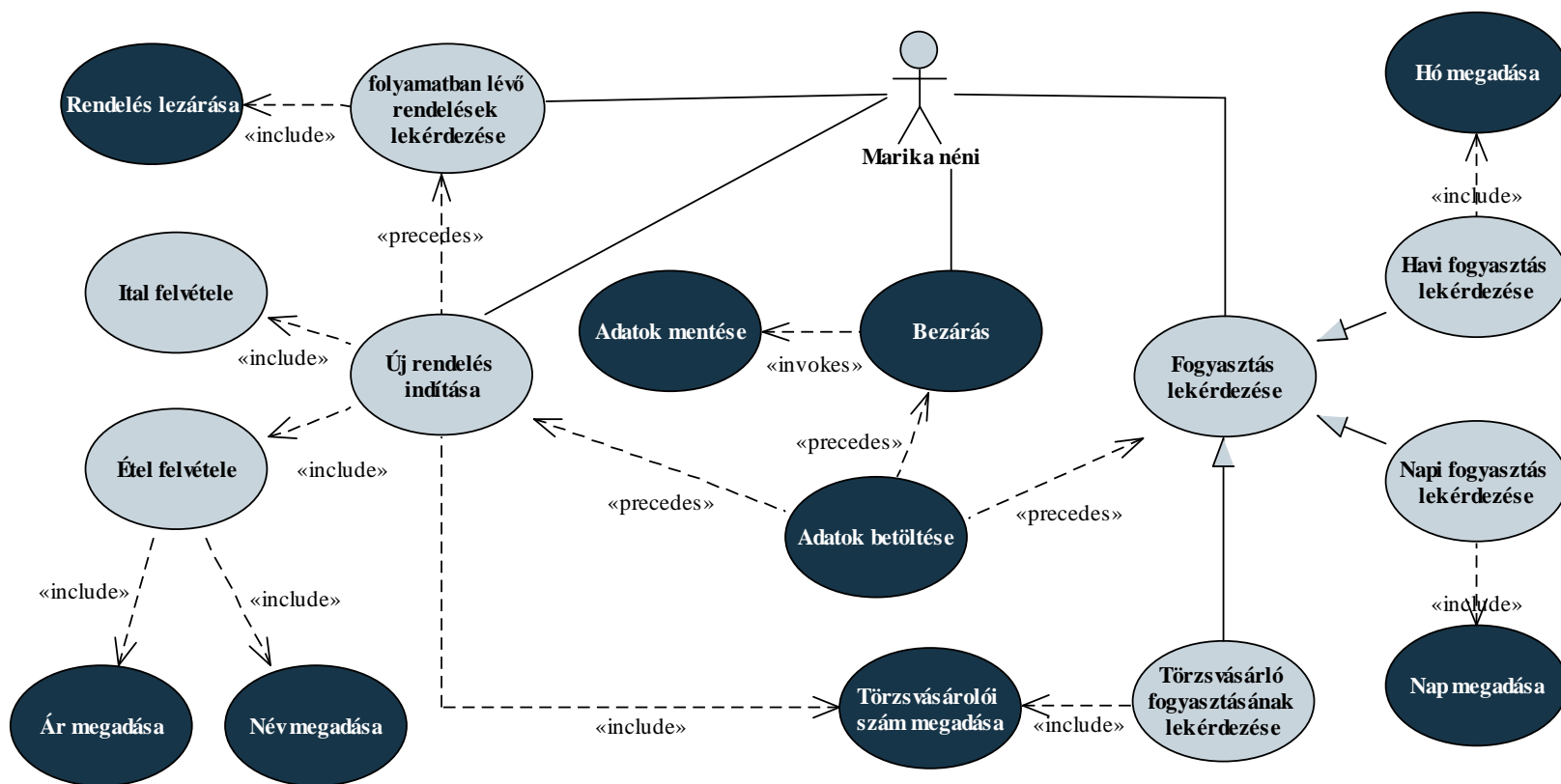
- a kávézóban 3 féle étel (hamburger, ufó, palacsinta), illetve 3 féle ital (tea, narancslé, kóla) közül lehet választani
- az ételek ezen belül különfélék lehetnek, amelyre egyenként lehet árat szabni, és elnevezni, az italok árai rögzítettek
- rendeléseket kell kezelnünk, amelyben tetszőleges tétel szerepelhet, illetve a rendelés tartozhat egy törzsvásárlóhoz
- lehetőségünk van utólagosan lekérdezni a függőben lévő rendeléseket, valamint napi, havi és törzsvásárlói számra összesített nettó/bruttó fogyasztást



# Objektumorientált tervezés

## 1. esettanulmány

- *Használati esetek:*



# Objektumorientált tervezés

## 1. esettanulmány

---

- *Szerkezeti tervezés* (0. fázis):
  - a programban rendeléseket kezelünk, amelyek tételekből állnak
  - a tételek a hamburger, ufó, palacsinta, kóla, narancs, tea, amelyek mind nagyon hasonlóak, csak néhány részletben térnek el
  - rendelések sorozatát kell kezelnünk a programban, amelyek száma folyamatosan bővül
  - a programot egy menün keresztül kezeljük, amely biztosítja a felhasználó felé a funkciókat, minden funkció ugyanazzal a rendelés sorozattal dolgozik

# Objektumorientált tervezés

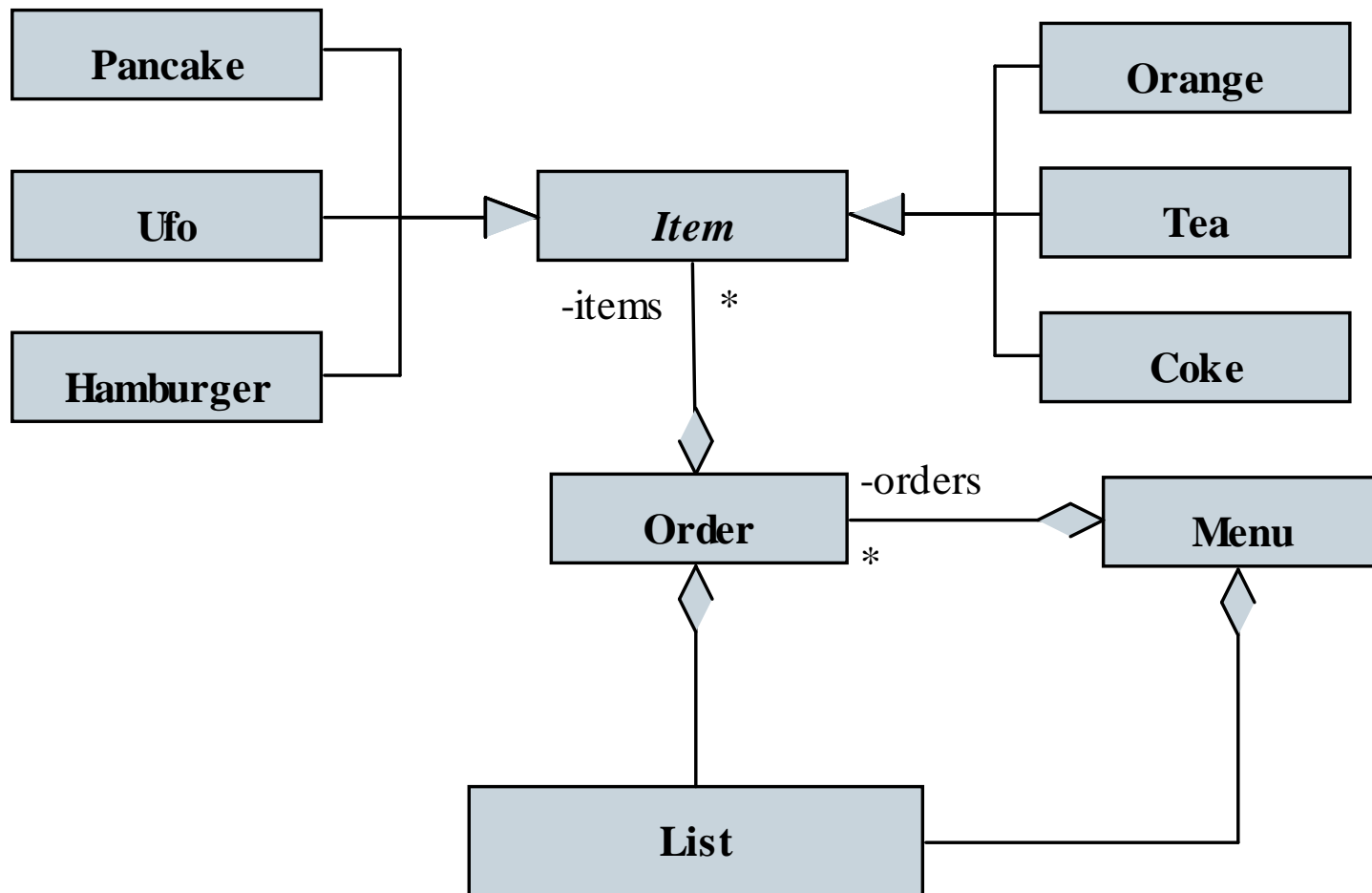
## 1. esettanulmány

---

- *Szerkezeti tervezés* (1. fázis):
  - a feladatban fellelhető tárgykörök a *menü*, a *rendelések sorozata*, a *rendelés*, valamint a *rendelés tételei* (italok, ételek)
  - *rendelés tételei* (**Item**):
    - hasonlóan viselkednek, ám némileg eltérően
    - ezért megvalósításban öröklődést használunk, specializáljuk a 3 ételt, illetve italt
  - *rendelés* (**Order**): tartalmazza a tételeket (mivel a rendelések száma változhat, ezért láncolt listát használ)
  - *menü* (**Menu**): tartalmazza a rendeléseket (láncolt listában)

# Objektumorientált tervezés

## 1. esettanulmány



# Objektumorientált tervezés

## 1. esettanulmány

---

- *Szerkezeti tervezés* (2. fázis):
  - *láncolt lista* (**List**):
    - külön megvalósítást igényel, sablonos típusként
    - kétszeresen láncolt, fejelemes, aciklikus reprezentáció
    - lehetőséget ad a beszúrásra (elején, végén, közben), törlésre, kiürítésre, és méret lekérdezésre
    - a listaelem (**ListItem**) tárolja az adatot és a két mutatót
    - a hibát kivétellel jelezzük, egy felsorolási típussal (**Exceptions**)
    - a lista bejárható, a bejáró (**Iterator**) a szabványos műveleteket tárolja

# Objektumorientált tervezés

## 1. esettanulmány

- *Szerkezeti tervezés* (2. fázis):
  - *rendelési tételek* (**Item**):
    - minden esetben ismert a név, a bruttó és a nettó ár
    - ám ezek csak az ételek esetén változnak
  - *rendelések* (**Order**):
    - adatai az azonosító (ez automatikus), a törzsvásárlói szám és a dátum, valamint, hogy folyamatban van-e
    - lehetőséget ad új elem felvételére, nettó/bruttó érték lekérdezésére
  - *menü* (**Menu**):
    - biztosítja a mentést/betöltést, valamint a menüfunkciókat

# Objektumorientált tervezés

## 1. esettanulmány

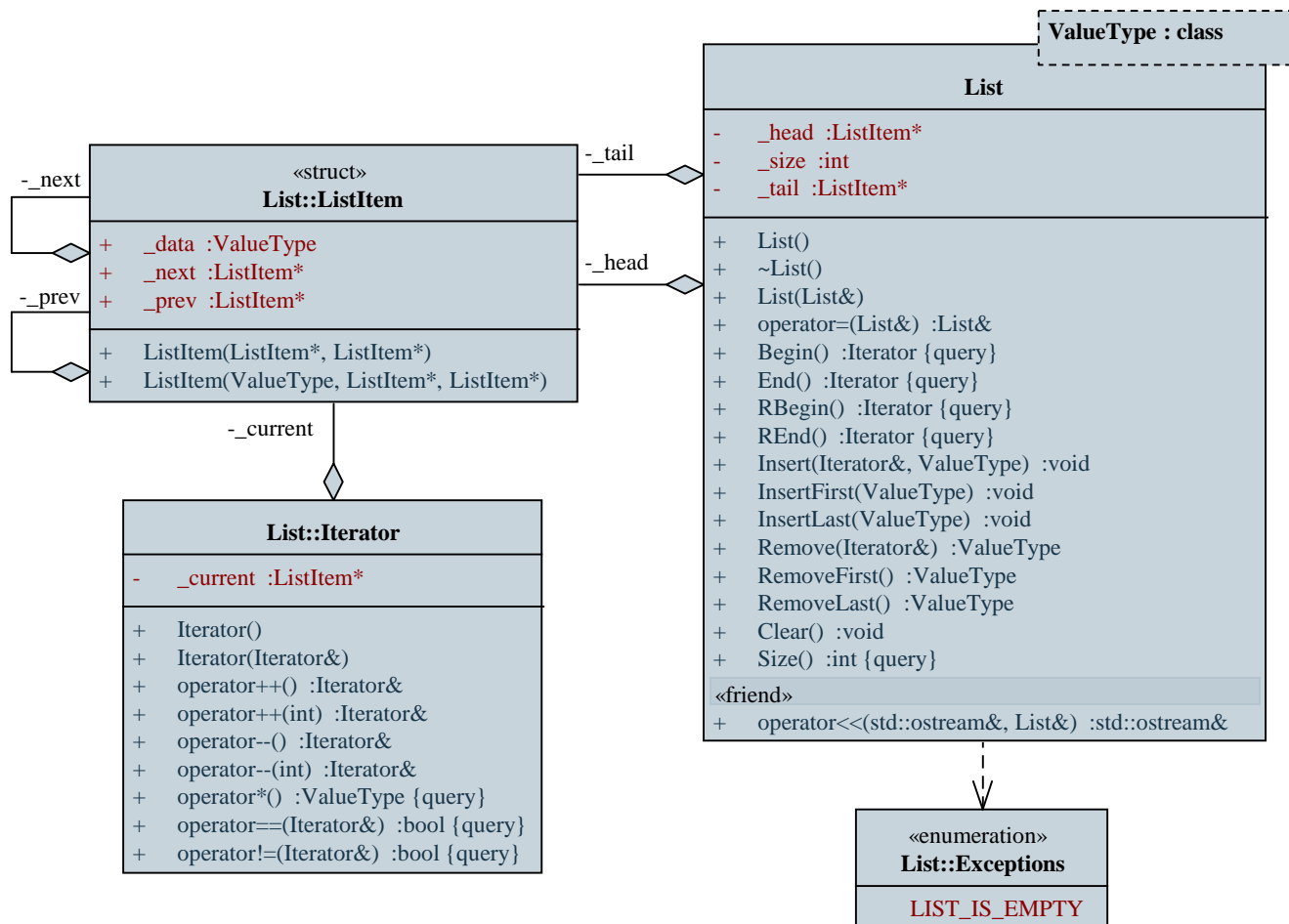
---

- *Szerkezeti tervezés* (3. fázis):
  - a cím szerinti hivatkozásokat mutatókon keresztül kezeljük
  - a listaelemet és a lista kivételeit beágyazott osztályként hozzuk létre, a listaelem egyszerűsége miatt lehet rekord (**struct**)
  - a lista megfelelő használatához szükséges destruktork, másoló konstruktor, értékadás művelet, kiíró operátor, indexelő operátor
  - a rendelési elem őssosztályban megvalósítunk egy virtuális destruktort

# Objektumorientált tervezés

## 1. esettanulmány

- *Szerkezeti tervezés (lista):*

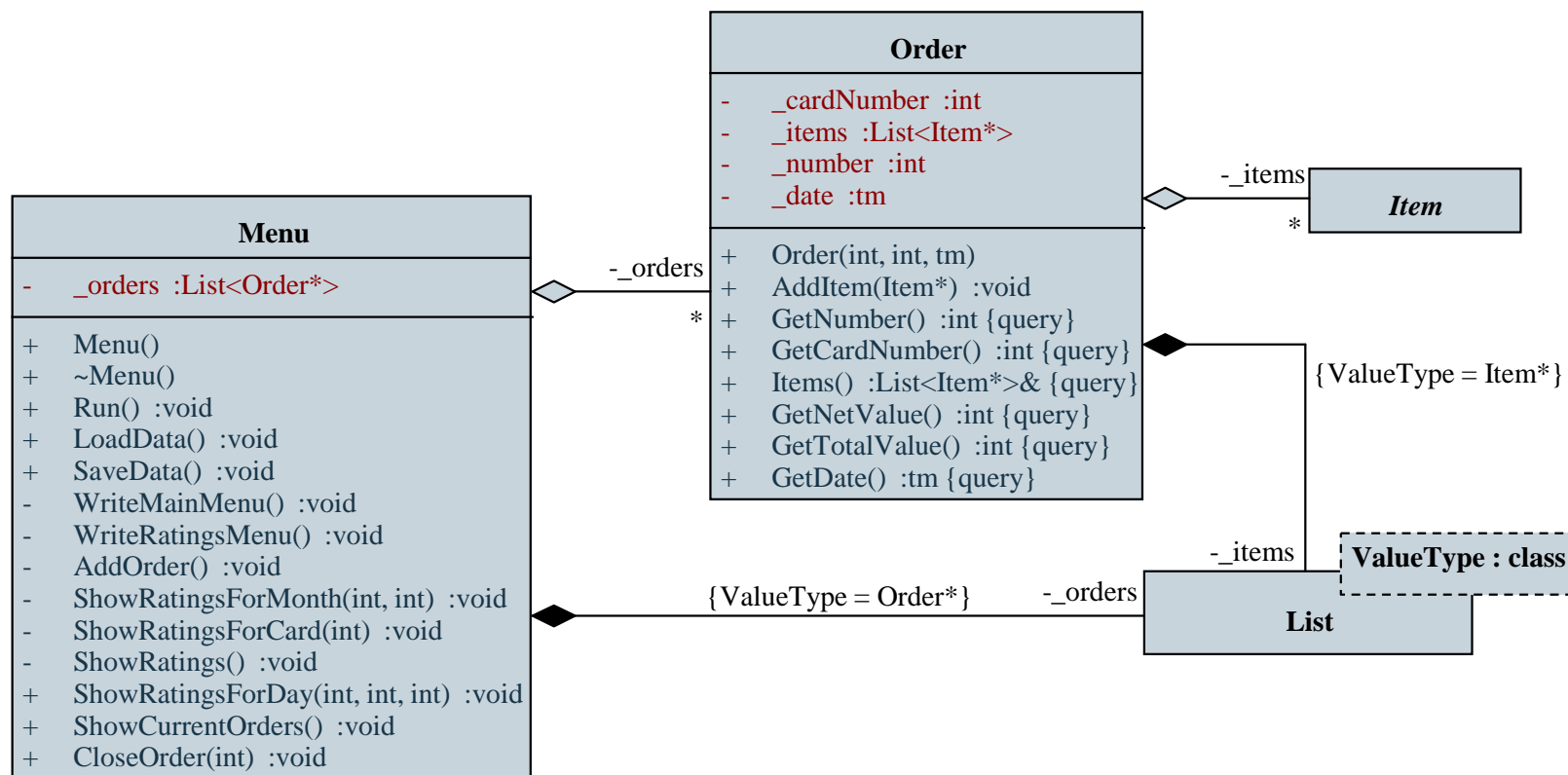




# Objektumorientált tervezés

## 1. esettanulmány

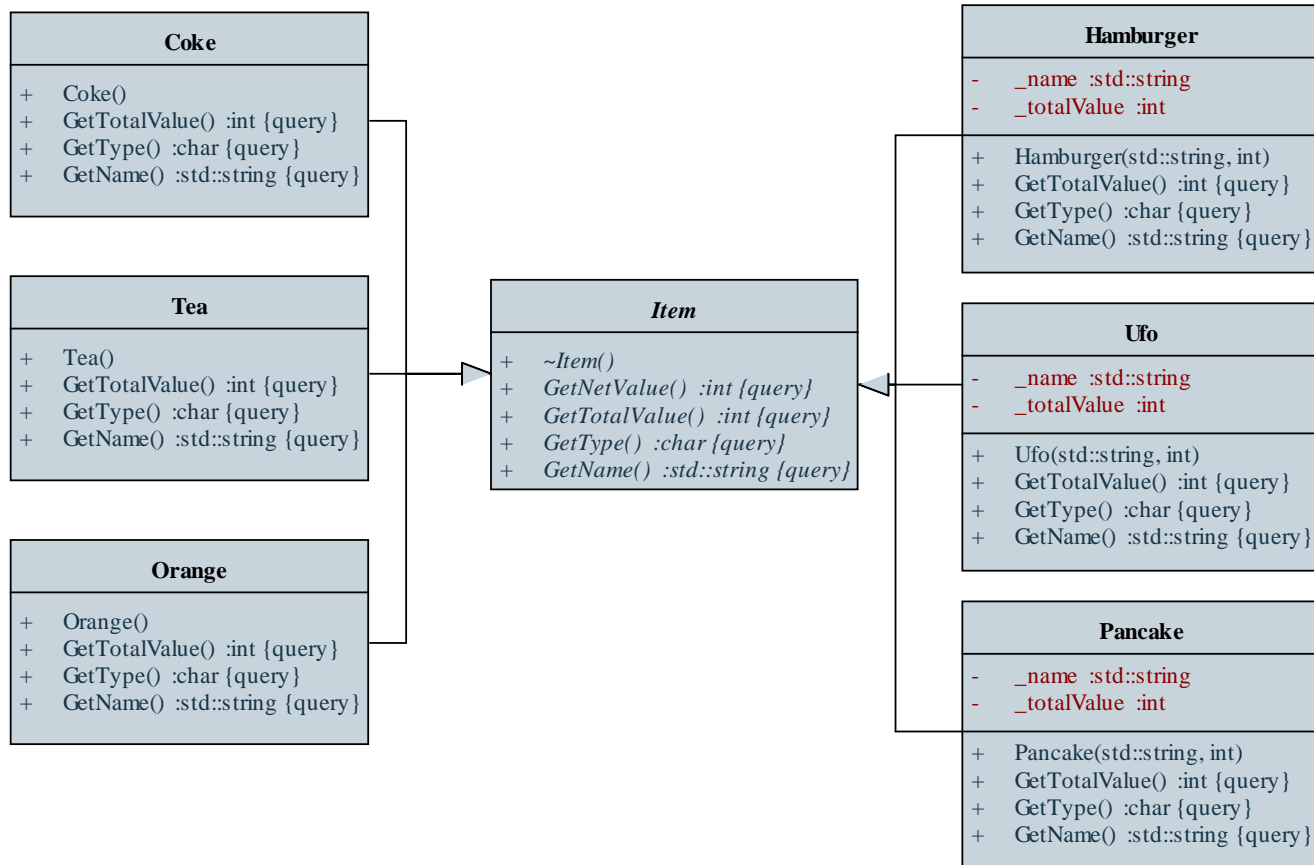
- *Szerkezeti tervezés (rendelések):*



# Objektumorientált tervezés

## 1. esettanulmány

- *Szerkezeti tervezés (tételek):*



# Objektumorientált tervezés

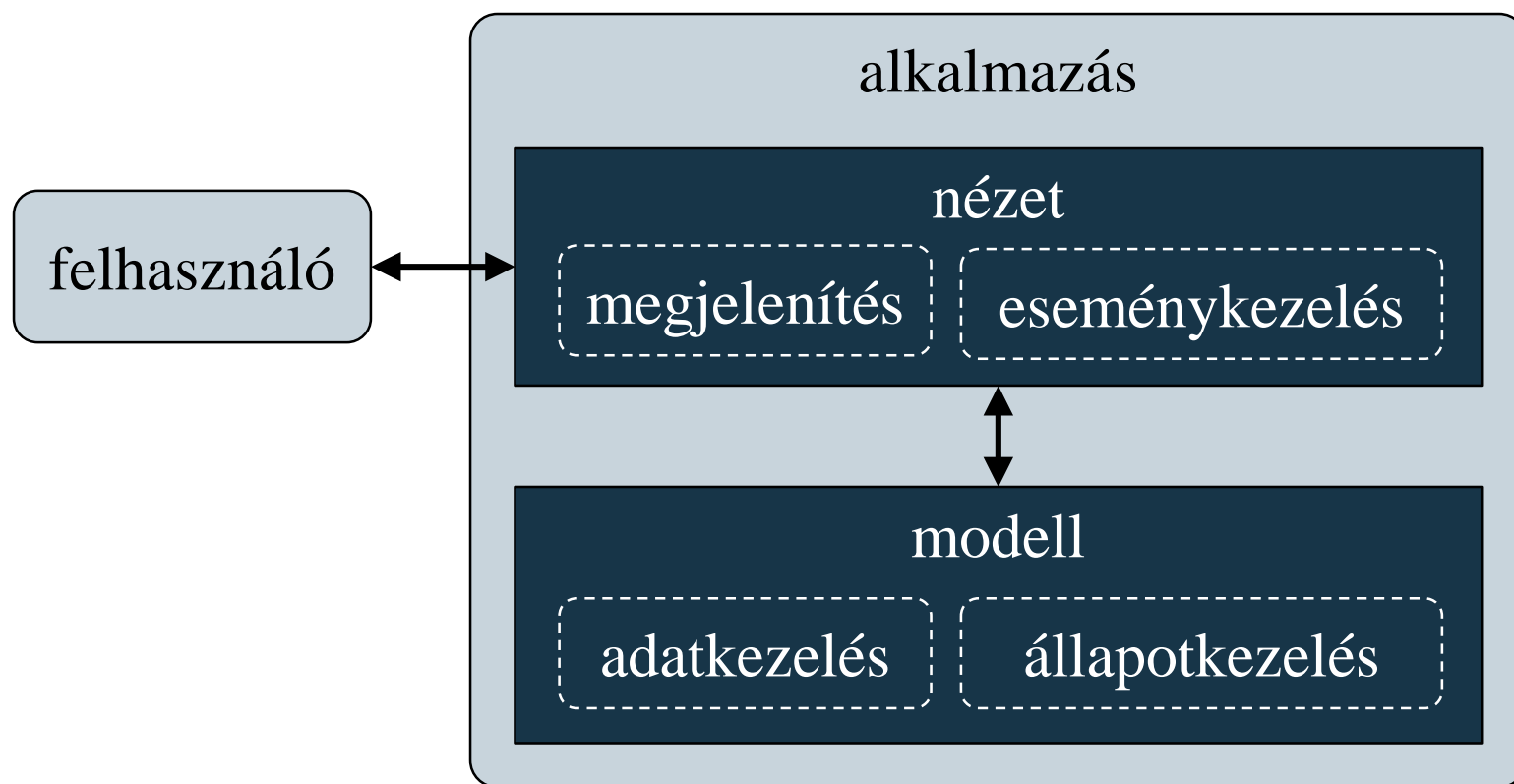
## A modell/nézet architektúra

- A programszerkezet felépítése akkor ideális, ha teljesen külön programegységbe tudjuk leválasztani a felhasználói felülettel kapcsolatos részeket a ténylegesen a feladat megoldását szolgáló funkcionálisról
- Ezt a felbontást követve jutunk el a *modell/nézet (MV, model-view)* architektúrához, amelyben
  - a *modell* tartalmazza a feladat végrehajtását szolgáló programegységeket, az állapotkezelést, valamint az adatkezelést, ezt nevezzük *alkalmazáslogikának*, vagy *üzleti logikának*
  - a *nézet* tartalmazza a grafikus felhasználói felület megvalósítását, a felület elemeit és az eseménykezelőket

# Objektumorientált tervezés

## A modell/nézet architektúra

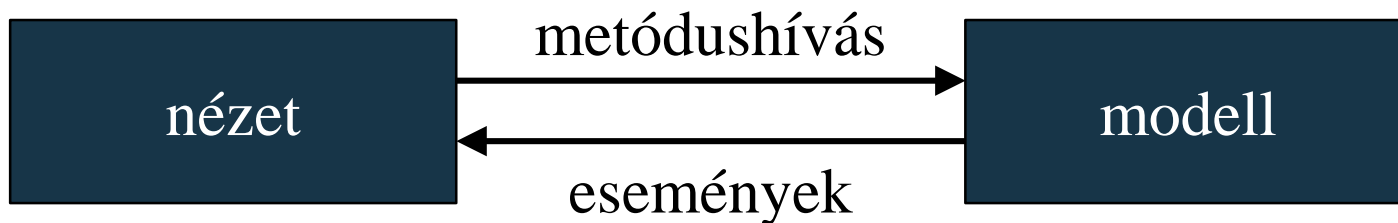
- a felhasználó a nézettel kommunikál, a modell és a nézet egymással



# Objektumorientált tervezés

## A modell/nézet architektúra

- A modell és a nézet kapcsolatát úgy kell megvalósítani, hogy
  - *a nézet ismerheti a modell felületét (interfészt), és hívhatja annak (publikus) műveleteit*
  - *a modellnek semmilyen tudomása sem lehet a nézetről, ezért nem hívhatja annak műveleteit, de eseményeken keresztül kommunikálhat vele*

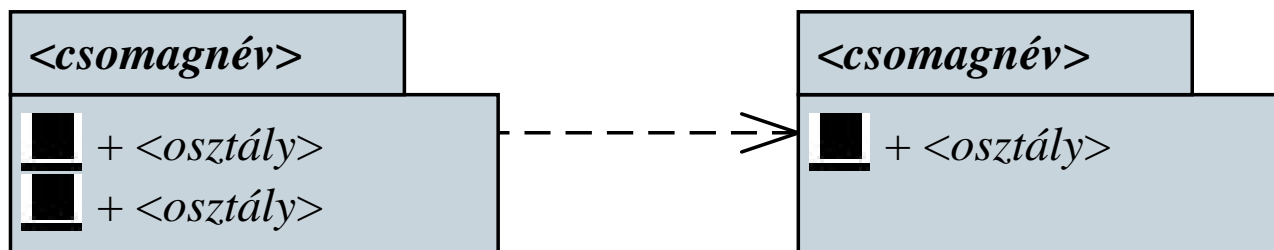


- A megvalósításban a nézet hivatkozhat a modellre (pontosabban a felületére)

# Objektumorientált tervezés

## Csomagdiagram

- A *csomagdiagram* (*package diagram*) célja a rendszer felépítése a logikai szerkezet mentén, azaz az egyes rétegek, illetve csomagok azonosítása és a csomagba tartozó osztályok bemutatása

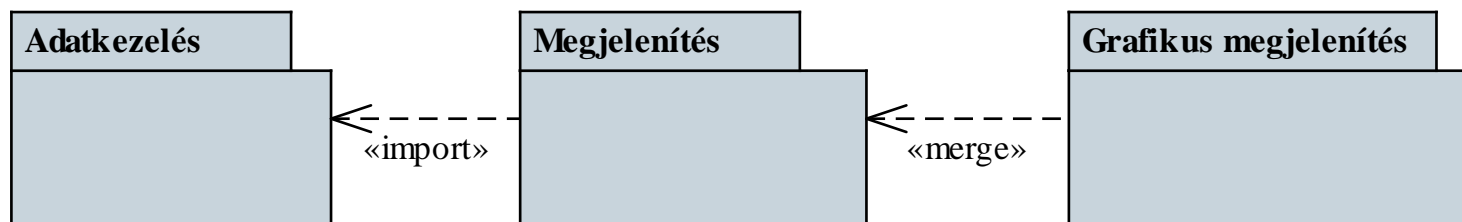


- a csomagok között is létrehozhatunk kapcsolatokat
  - az osztályok közötti kapcsolatok érvényesek: függőség, asszociáció, általánosítás, megvalósítás

# Objektumorientált tervezés

## Csomagdiagram

- *használat* (**use**): a csomag felhasznál egy másikat
- *beágyazás* (**nesting**): a csomag egy másiknak
- *importálás* (**import**): a csomag betölti a másikat
- *összeillesztés* (**merge**): a csomag tartalmazza a másik teljes funkcionalitását



- Amennyiben egy réteg több csomagból is áll, akkor azokat beágyazott csomagként jelölhetjük a diagramban

# Objektumorientált tervezés

## 2. esettanulmány

2. *esettanulmány*: Készítsünk egy *Memory* kártyajátékot, amelyben két játékos küzd egymás ellen. A játékmezőn kártyapárok találhatóak, és a játékosok feladata ezek megtalálása.

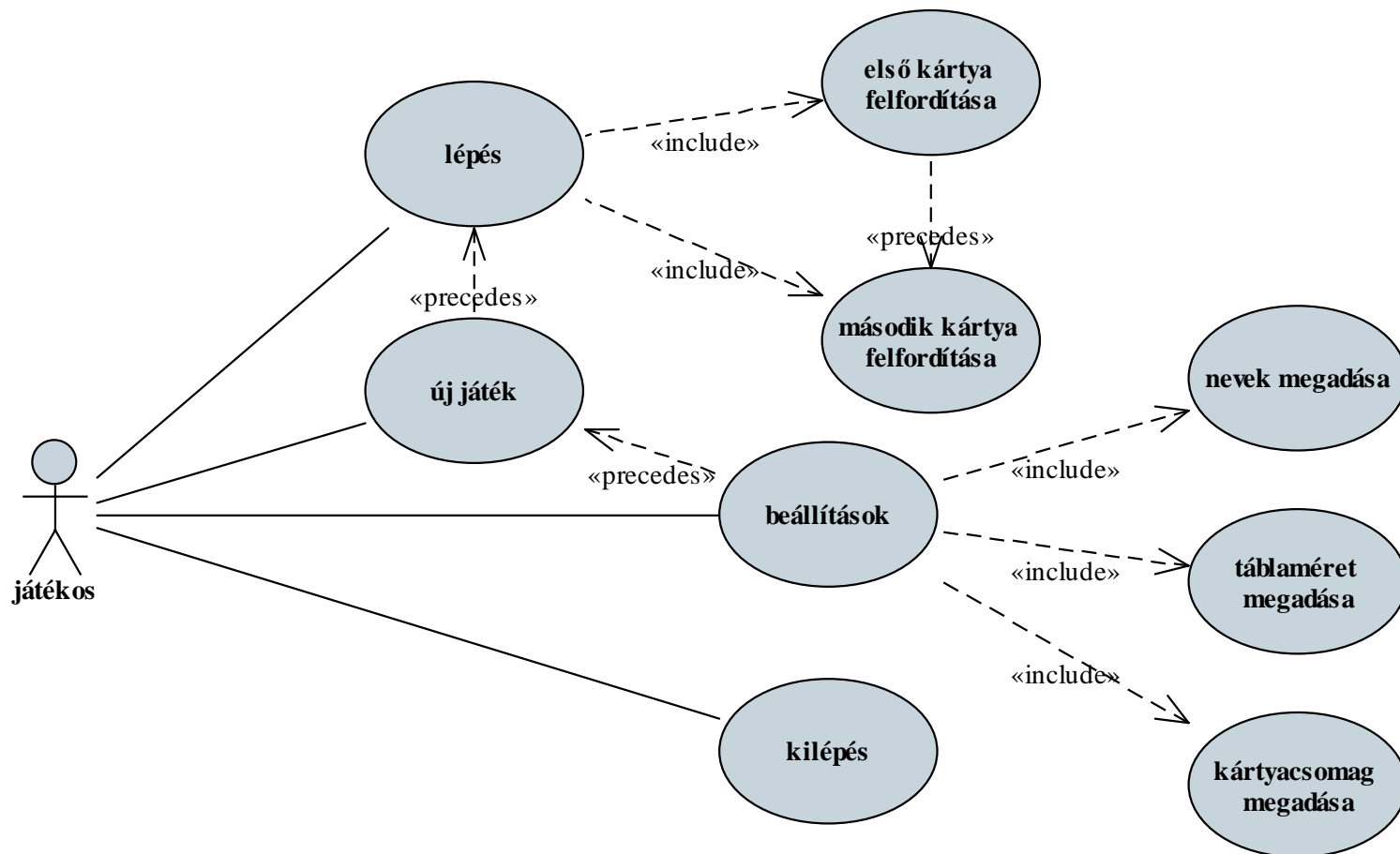
- a játékban választhatunk kártyacsomagot, a játékosok megadhatják neveiket, valamint a játék méretét (kártyák száma)
- a játékosok felváltva lépnek, minden lépésben felfordíthatnak két kártyát, amennyiben egyeznek, úgy felfordítva maradnak és a játékos ismét léphet, különben 1 másodperc múlva visszafordulnak
- a játékot az nyeri, aki több kártyapárt talált meg



# Objektumorientált tervezés

## 2. esettanulmány

- *Használati esetek:*



# Objektumorientált tervezés

## 2. esettanulmány

- *Szerkezeti tervezés:*
  - a játékot kétrétegű architektúrában valósítjuk meg
  - a modell tartalmazza:
    - magát a játékot, amit egy kezelőosztály felügyel (**GameManager**), valamint hozzá segédosztályként a játékost (**Player**)
    - a kártyacsomagokat (**CardPack**)
  - a nézet tartalmazza:
    - a játék főablakát (**MainWindow**), amely tartalmaz egy menüt és egy státuszsort
    - a beállítások segédablakát (**ConfigurationDialog**)

# Objektumorientált tervezés

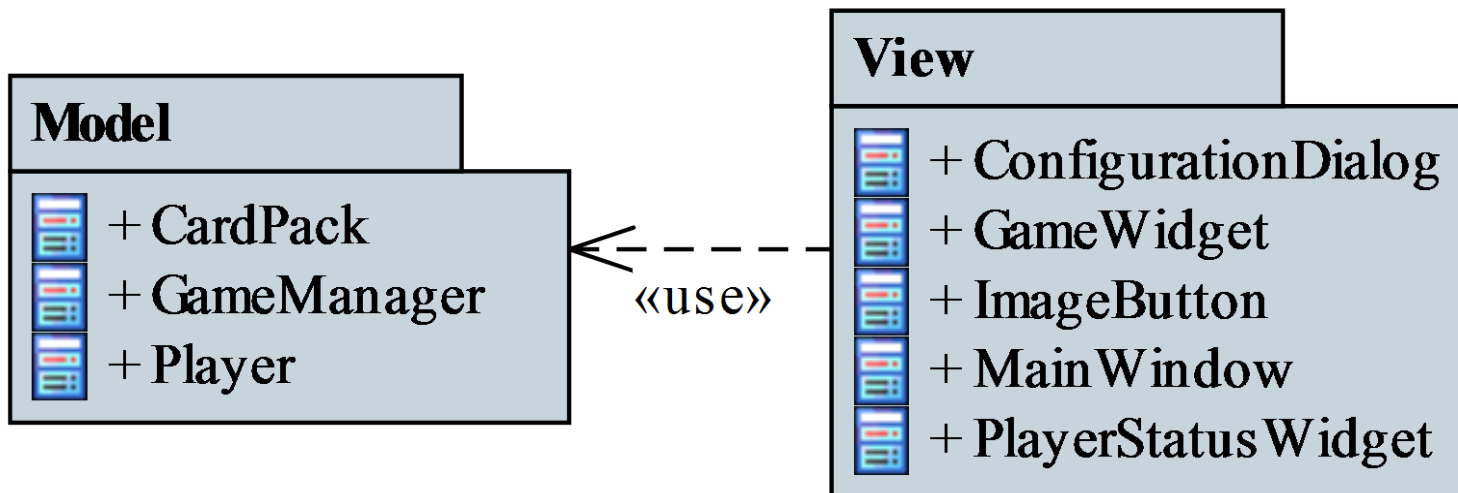
## 2. esettanulmány

- a játékelületet megjelenítő vezérlőt (**GameWidget**), amely tartalmazza a játékmezővel kapcsolatos tevékenységeket
- ehhez segédosztályként a felhasználói információkat kiíró vezérlőt (**PlayerStatusWidget**, ezt előléptetett vezérlővel állítjuk be a felülettervezőben), valamint a képet megjeleníteni tudó egyedi gombot (**ImageButton**)
- a nézet a modell publikus műveleteit hívja, és eseményeket is kaphat tőle
- egy csomag kártyát erőforrásként csatolunk az alkalmazáshoz (**packs.qrc**), hogy mindig legyen legalább egy csomag kártya

# Objektumorientált tervezés

## 2. esettanulmány

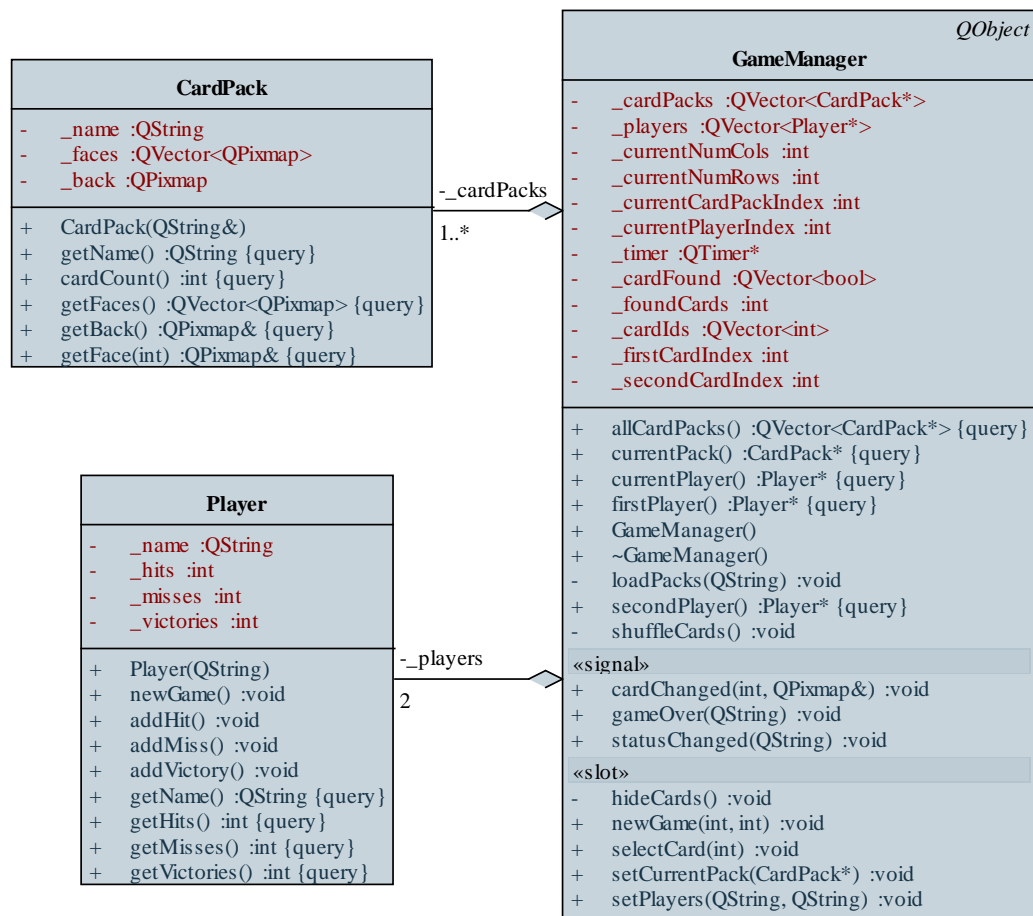
- *Szerkezeti tervezés (csomagok):*



# Objektumorientált tervezés

## 2. esettanulmány

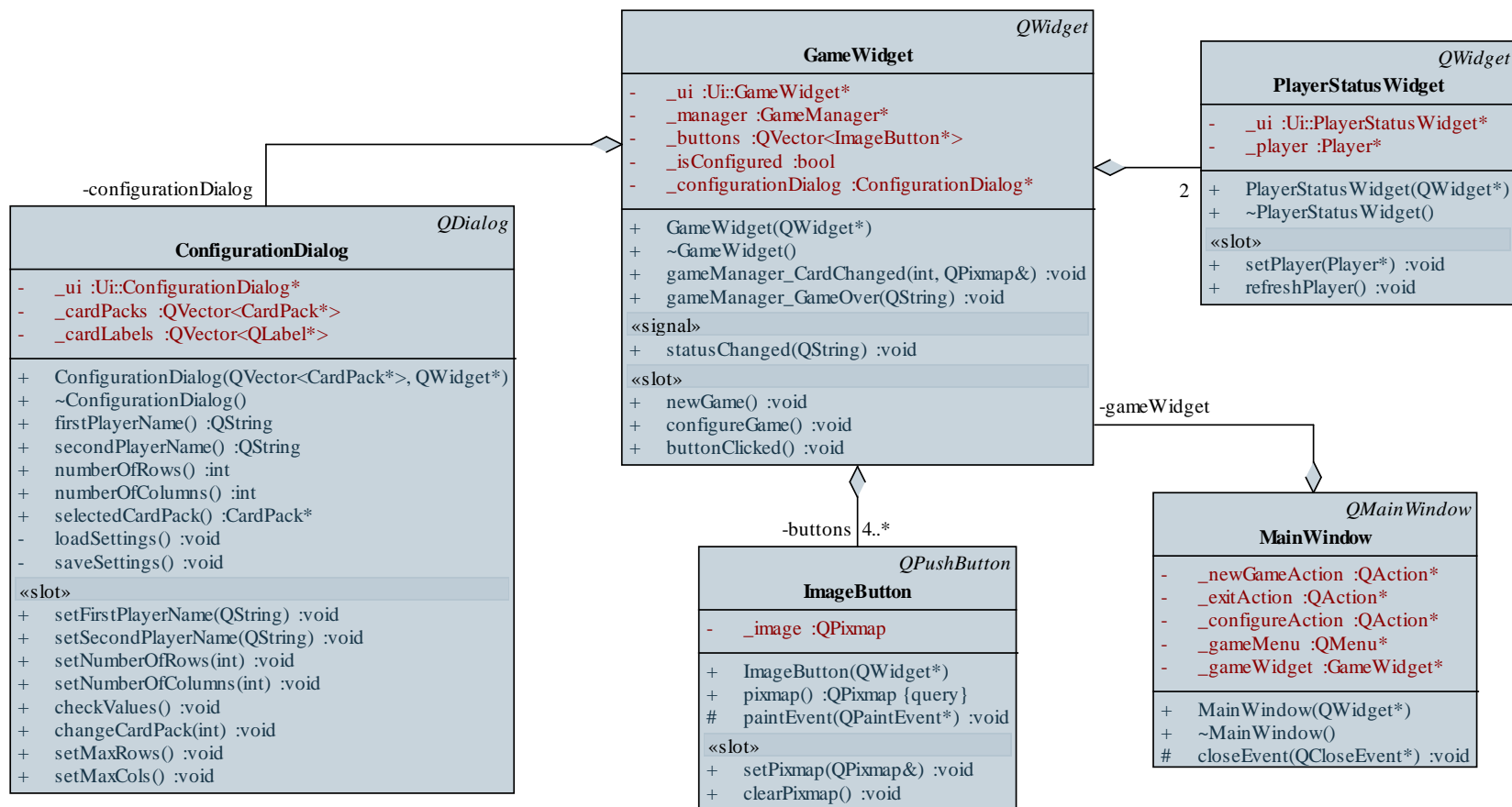
- *Szerkezeti tervezés (modell):*



# Objektumorientált tervezés

## 2. esettanulmány

- Szerkezeti tervezés (nézet):



# Objektumorientált tervezés

## A szoftverrendszer

- Szoftvernek nevezzük a program(ok), dokumentáció(k), konfiguráció(k), valamint adatok együttese
  - mivel a megoldandó feladatok összetettek lehetnek, a megoldást nem feltétlenül egy program, hanem több program tudja megadni
  - a végrehajtás során ezek a programok egymással kommunikálnak (adatot cserélnek)
- Egymással kommunikáló programok álkotta szoftvereket nevezzük *szoftverrendszernek (software system)*
  - a rendszerben jelen lévő programokat nevezzük a rendszer *komponenseinek (component)*

# Objektumorientált tervezés

## Komponensek

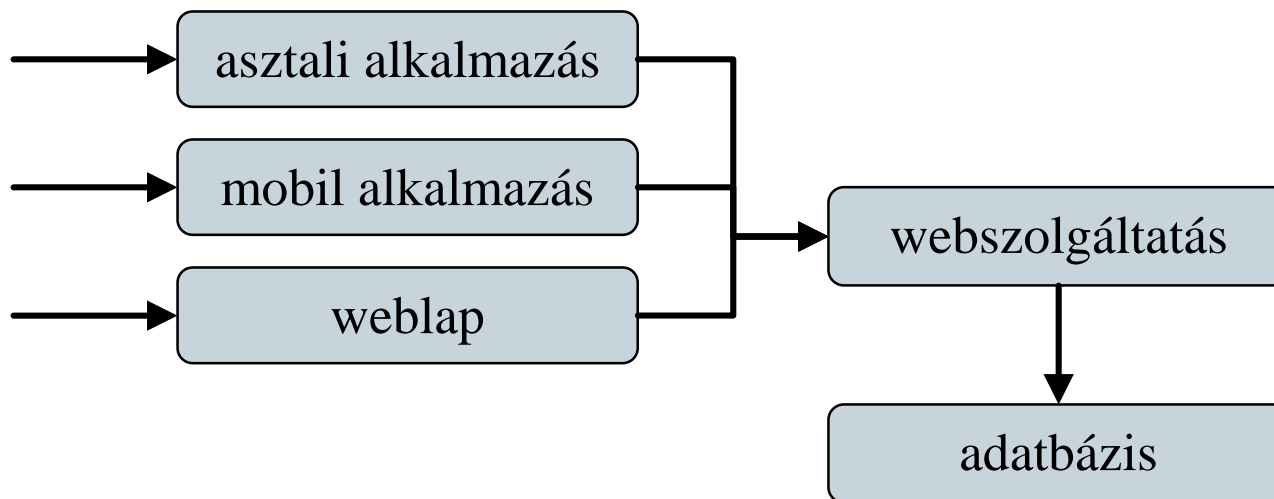
- A szoftver komponens egy adott funkcionalitásért felelő, fizikailag elkülönülő része a rendszernek
  - önállóan (újra)felhasználható, telepíthető
  - belső működése rejtett, a kapcsolatot megfelelő *felületen (interface)* keresztül teremti meg
  - szolgáltathat olyan funkcionalitást, amelyet más komponensek használnak fel, ehhez tartozik egy *szolgáltatott felület (provided interface)*
  - felhasználhat más komponenseket, amelyek funkcionalitását egy *elvárt felületen (required interface)* keresztül érhetik el



# Objektumorientált tervezés

## Komponensek

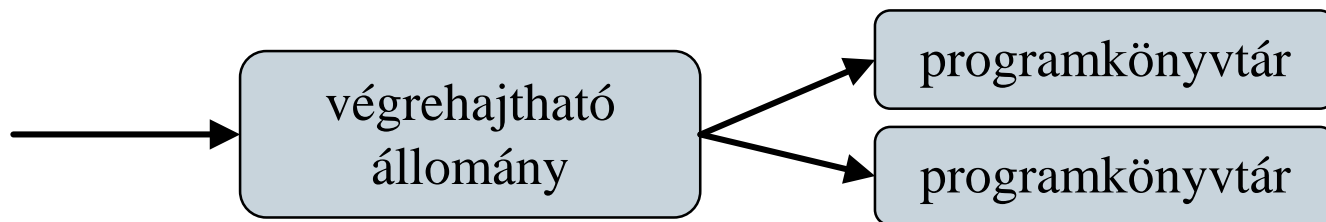
- Egy szoftverrendszerben számos komponens található, pl.
  - mobil alkalmazás, asztali alkalmazás, weblap (biztosítják a kapcsolatot a felhasználóval)
  - webszolgáltatás (gondoskodik az adatok továbbításáról)
  - adatbázis (gondoskodik az adatok megfelelő tárolásáról)



# Objektumorientált tervezés

## Komponensek

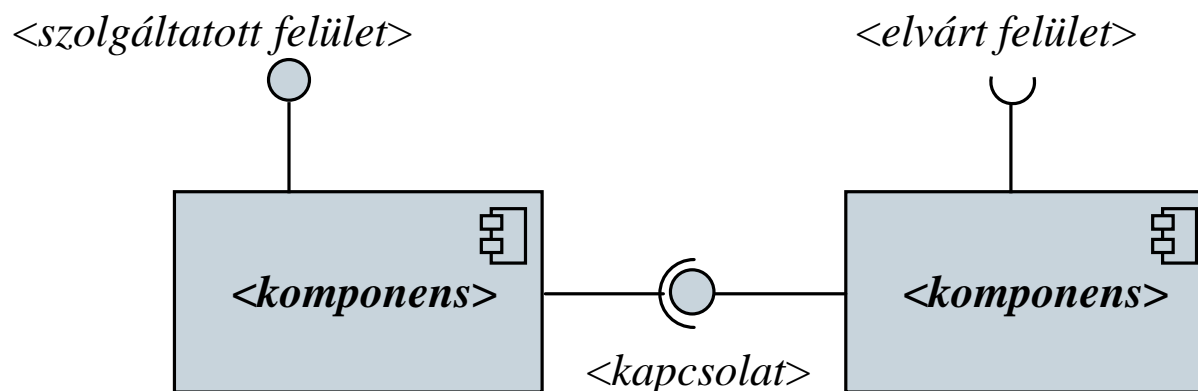
- Egy program is felbontható komponensekre, amennyiben egyes részeit újrafelhasználhatóvá szeretnénk tenni
- Egy program komponensei lehetnek:
  - végrehajtható állomány (*executable*), amely biztosítja a belépési pontot az alkalmazásba
  - programkönyvtár (*library*), amely adott funkciók gyűjteménye (nem végrehajtható), objektumorientált környezetben osztályok gyűjteménye (*class library*)



# Objektumorientált tervezés

## Komponensdiagram

- A szoftverrendszer komponenseit *UML komponensdiagram* (*component diagram*) segítségével ábrázolhatjuk
  - ismerteti a rendszer komponenseit, a szolgáltatott/elvárt felületeket és a közöttük fennálló kapcsolatokat (*connector*)

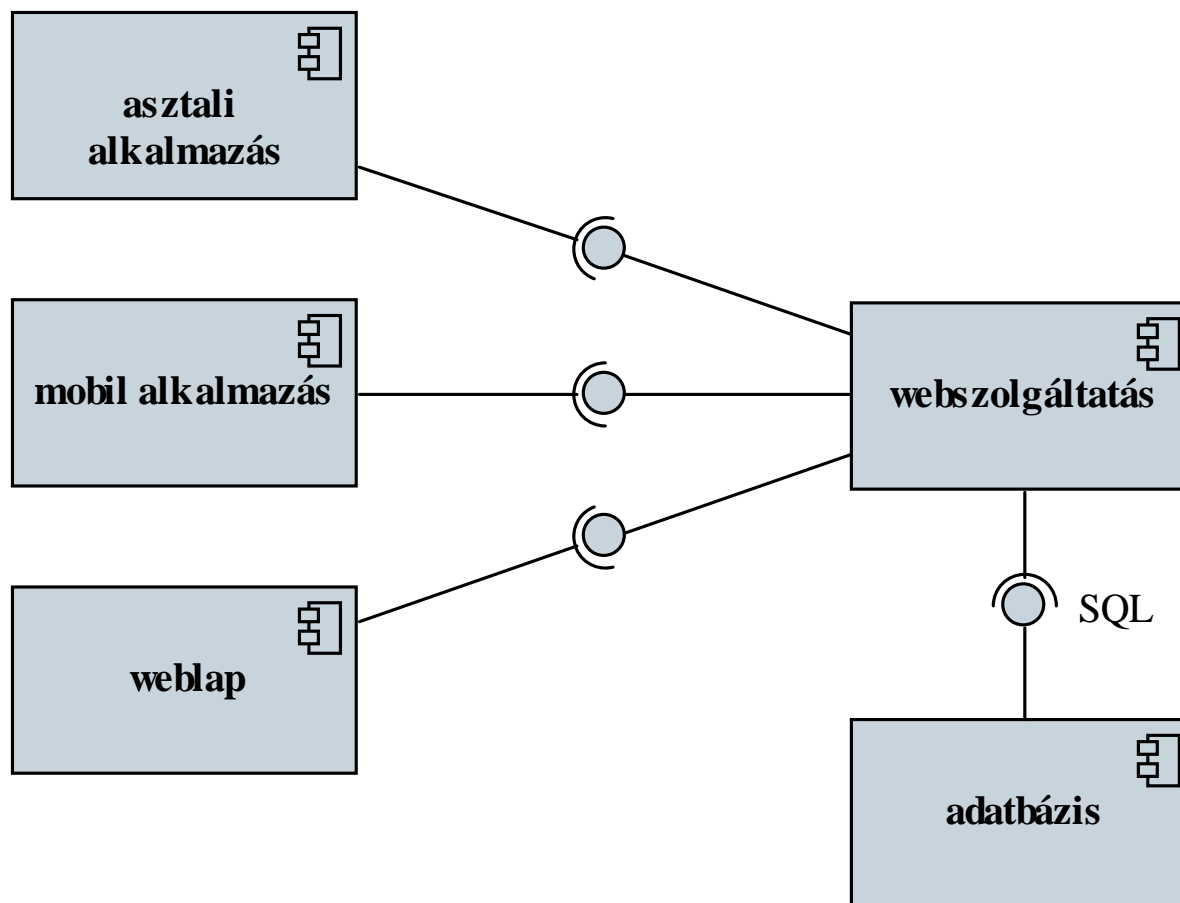


- a komponens diagramnak osztálydiagram elemeket is elhelyezhetünk (pl. felület külön megjeleníthető)

# Objektumorientált tervezés

## Komponensdiagram

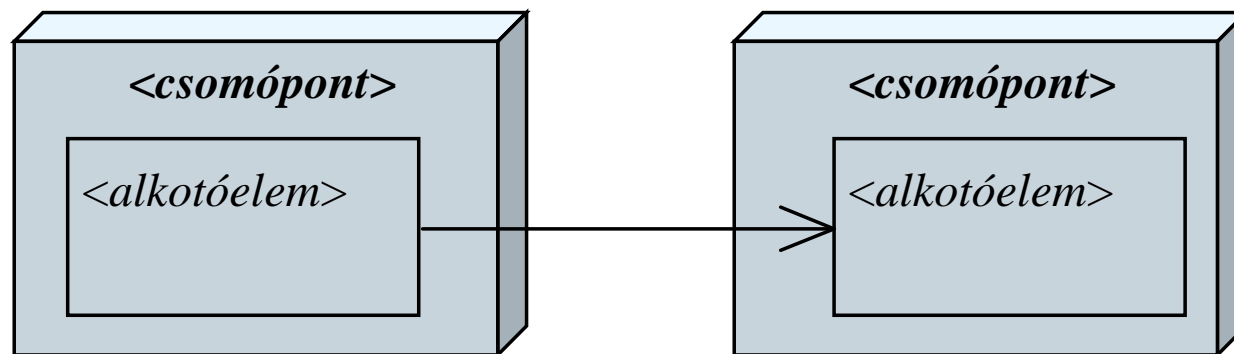
- Pl.:



# Objektumorientált tervezés

## Telepítési diagram

- A szoftverrendszerek komponensei akár különböző hardver eszközökre is kihelyezhetőek, amelyeken interakcióba lépnek a környezetükkel (más szoftverekkel)
- A szoftverrendszert kihelyezési és környezeti szempontból az *UML telepítési diagram (deployment diagram)* ábrázolja
  - ismerteti azon *csomópontokat (node)*, amelyekre az egyes *alkotóelemei (artifact)* találhatóak



# Objektumorientált tervezés

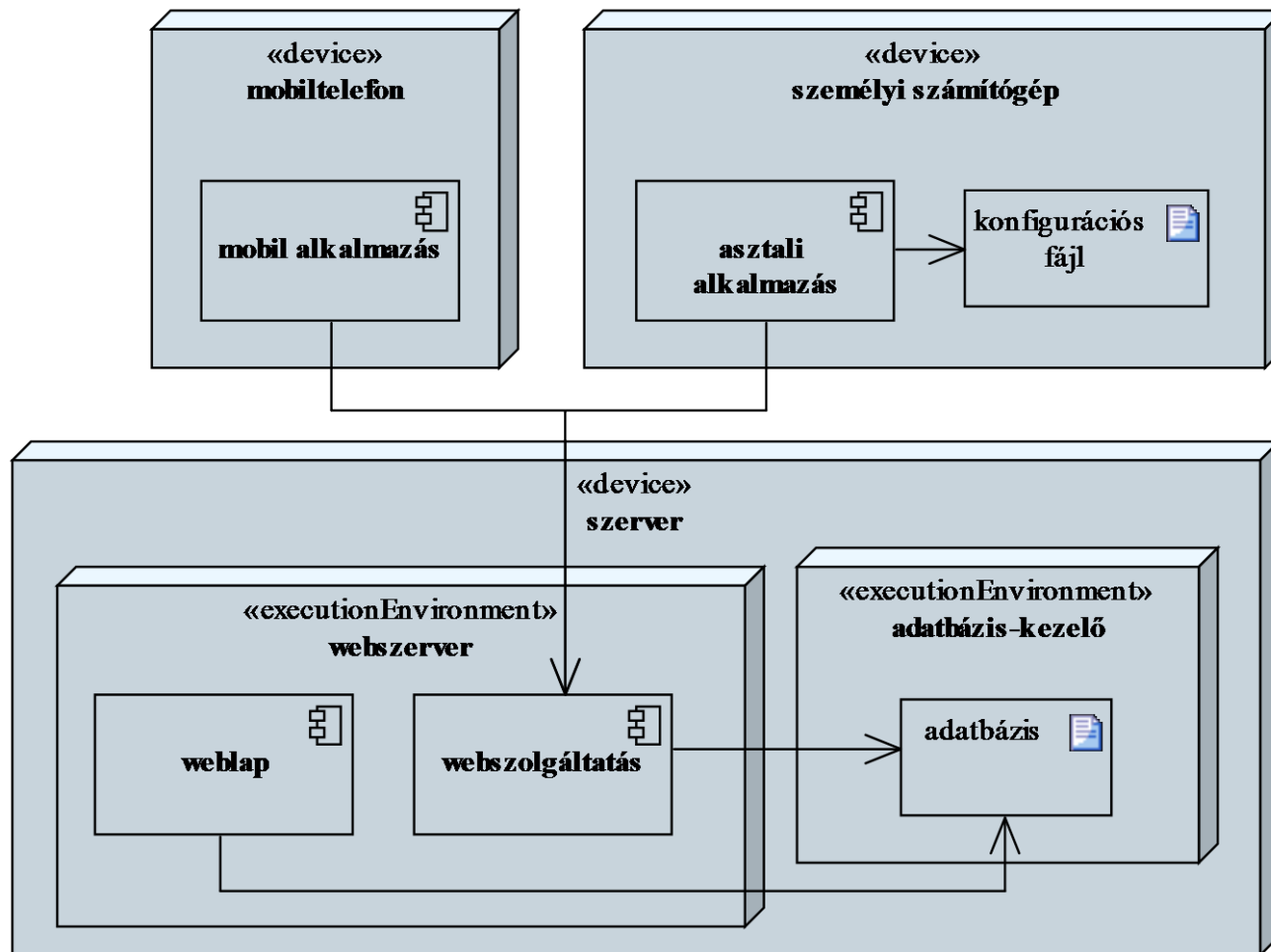
## Telepítési diagram

- A rendszer alkotóeleme lehet bármilyen, fizikailag elkülönülő tartozéka a szoftvernek
  - pl. mobil alkalmazás, weblap, kódfájl, adatfájl, adatbázis, konfigurációs fájl
  - a komponenseket jelölhetjük komponensként
- A rendszer csomópontja lehet:
  - egy *hardver eszköz (device)*, amelyen futtatjuk a szoftvert  
pl. mobiltelefon, szerver gép
  - egy *végrehajtási környezet (execution environment)*, amely biztosítja szoftverek futtatását, pl. webszerver, virtuális gép, adatbázis-kezelő

# Objektumorientált tervezés

## Telepítési diagram

- Pl.:



# Objektumorientált tervezés

## Adatformátumok

---

- A szoftverrendszer tervezése (*system design*) mellett foglalkoznunk kell a rendszer által kezelt adatok kezelésének módjával, formátumának meghatározásával, ez az adat tervezés (*data design*)
  - minden, a szoftver (vagy komponensei) számára bemenetként, vagy kimenetként szolgáló adat formátumát, felépítését meg kell adnunk (pl. adatfájl, adatbázis, konfigurációs fájl, felhasználó által letölthető adatok)
  - összetett adatok esetén támaszkodhatunk létező formátumokra (pl. CSV, XML, JSON), vagy létrehozhatunk egyedi formátumot



# Objektumorientált tervezés

## 1. esettanulmány

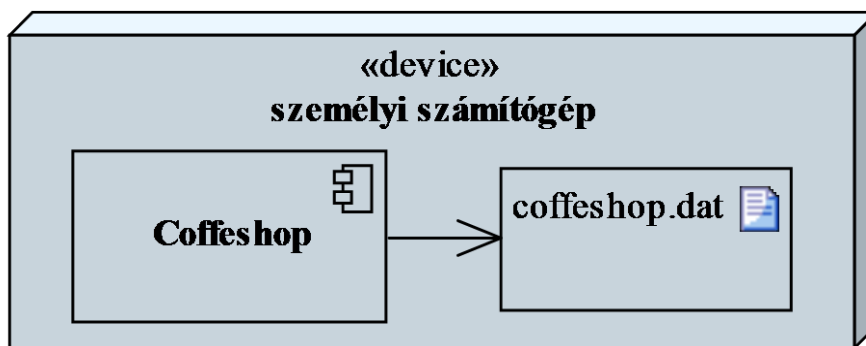
*1. esettanulmány:* Készítsük el Marika néni kávézójának eladási nyilvántartását végigkövető programot.

- a kávézóban 3 féle étel (hamburger, ufó, palacsinta), illetve 3 féle ital (tea, narancslé, kóla) közül lehet választani
- az ételek ezen belül különfélék lehetnek, amelyre egyenként lehet árat szabni, és elnevezni, az italok árai rögzítettek
- rendeléseket kell kezelnünk, amelyben tetszőleges tétel szerepelhet, illetve a rendelés tartozhat egy törzsvásárlóhoz
- lehetőségünk van utólagosan lekérdezni a függőben lévő rendeléseket, valamint napi, havi és törzsvásárlói számra összesített nettó/bruttó fogyasztást

# Objektumorientált tervezés

## 1. esettanulmány

- *Tervezés* (telepítés):
  - a program egy komponensben valósul meg, egy személyi számítógépen fog futni
  - a program közvetlenül az operációs rendszeren fut, nincs külön igénye a végrehajtási környezetre
  - a program az adatokat egy fájlban (**coffeshop.dat**) szöveges formában fogja tárolni



# Objektumorientált tervezés

## 1. esettanulmány

- *Tervezés* (adatformátum):
  - a fájlban rendelések következnek egymás után, minden rendelésnél adott az azonosító, a dátum, a törzsvásárolói kártya száma (vagy 0, amennyiben nincs) és a tételek száma
  - a rendelés utána felsoroljuk a tételeket, minden tételnél megadjuk a típust (ehhez elég egy karakter)
  - amennyiben a tétel egy étel, akkor rögzítjük a pontos nevet, illetve a bruttó árat
  - CSV formátumnak megfelelően a fájlban a tartalmi elemeket (rendelés, tétel) sortörés választja el, a soron belül a tartalmat pontosvessző segítségével választjuk el

# Objektumorientált tervezés

## 1. esettanulmány

- *Tervezés* (adatformátum):
  - a fájl szerkezetének sémája:  
`<rendelés azonosító>;<dátum>;<törzsv. szám>;  
<tételek száma>  
<típus: h/u/p/t/n/k>;<étel neve>;<étel ára>  
<típus: h/u/p/t/n/k>;<étel neve>;<étel ára>  
...  
<rendelés azonosító>;<dátum>;<törzsv. szám>;  
<tételek száma>  
...  
• pl.:  
184601;2015-11-11;73;2  
h;béke;800  
t`

# Objektumorientált tervezés

## 2. esettanulmány

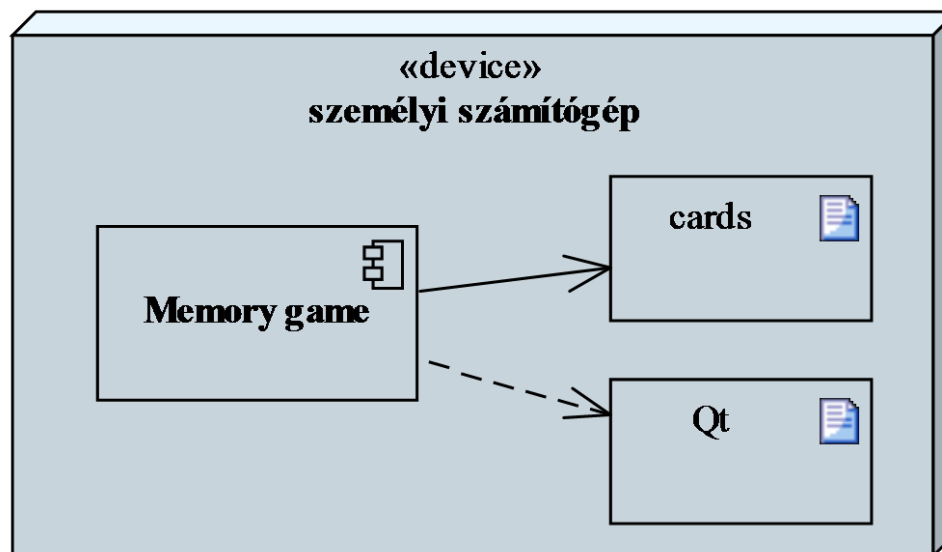
2. *esettanulmány*: Készítsünk egy *Memory* kártyajátékot, amelyben két játékos küzd egymás ellen. A játékmezőn kártyapárok találhatóak, és a játékosok feladata ezek megtalálása.

- a játékban választhatunk kártyacsomagot, a játékosok megadhatják neveiket, valamint a játék méretét (kártyák száma)
- a játékosok felváltva lépnek, minden lépésben felfordíthatnak két kártyát, amennyiben egyeznek, úgy felfordítva maradnak és a játékos ismét léphet, különben 1 másodperc múlva visszafordulnak
- a játékot az nyeri, aki több kártyapárt talált

# Objektumorientált tervezés

## 2. esettanulmány

- *Tervezés* (telepítés):
  - a program egy komponensben valósul meg, egy személyi számítógépen fog futni, és igényli a QT keretrendszer meglétét
  - a program a kártyacsomagok képeit külön tárolja



# Objektumorientált tervezés

## 2. esettanulmány

- *Tervezés* (adatformátum):
  - minden kártyacsomagnak van egy neve, valahány lapja, illetve egy hátoldala, ezeket képfájlban, PNG formátumban tároljuk
  - a kártyacsomagokat könyvtáranként helyezük el, minden könyvtárban található egy szöveges fájl (**name.txt**), amely tartalmazza a csomag nevét
  - a hátlapot egy fájlban (**back.png**) tároljuk, ez sosem változik
  - az előlapok fájljait sorszámozzuk (**<sorszám>.png**), és feltételezzük, hogy minden fájl más képet tartalmaz

# Objektumorientált tervezés

## 3. esettanulmány

3. *esettanulmány*: Készítsük el egy utazási ügynökség apartmanokkal foglalkozó rendszerét.

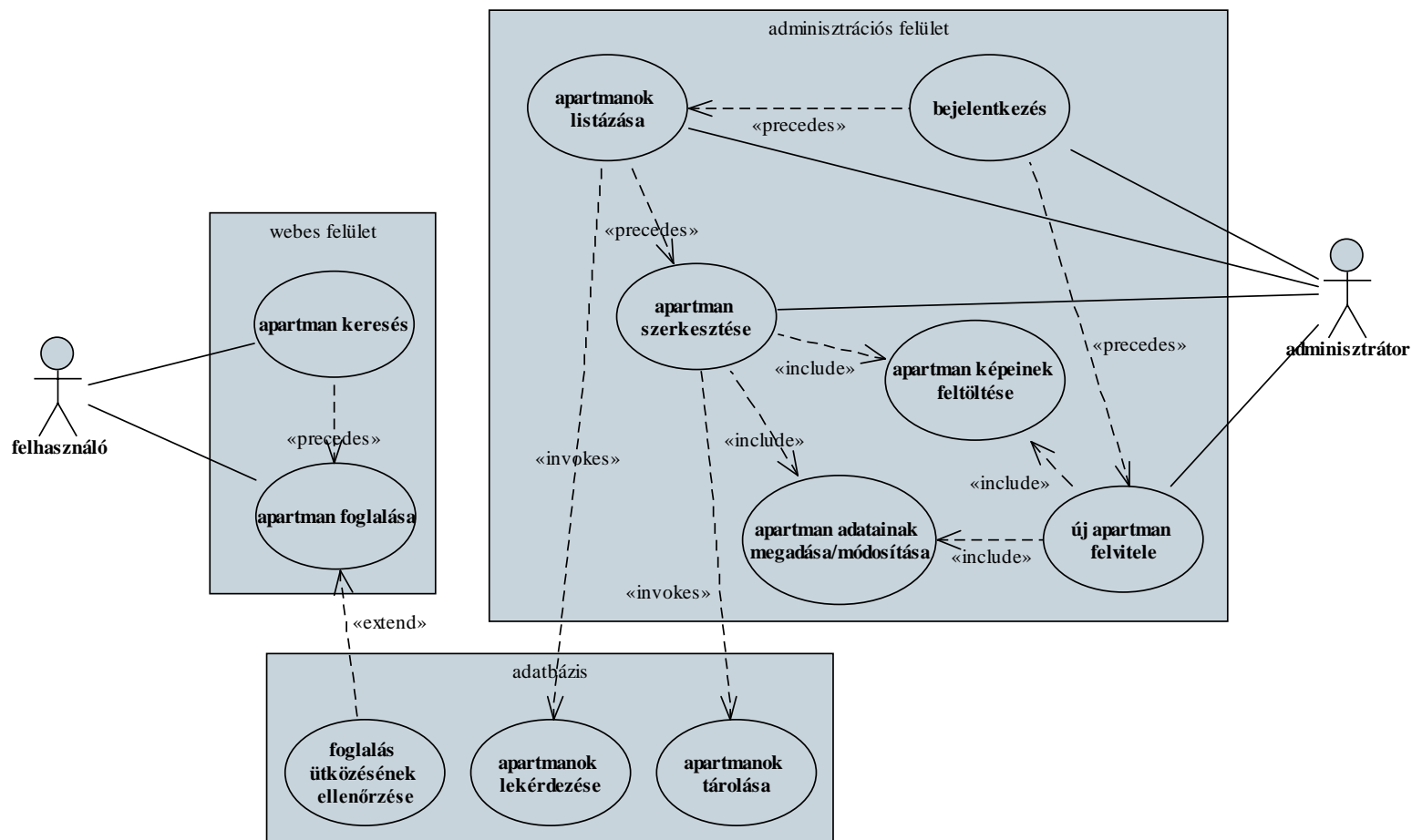
- az apartmanok épületekben találhatóak, amelyek városokban találhatóak
- az épületek különböző adatokkal (leírás, szolgáltatások, pontos hely, tengerpart távolság, ...), valamint képekkel rendelkeznek
- a felhasználók egy webes felületen keresztül foglalhatnak apartmanokat (adataik, valamint a foglalás időpontja megadásával), amelyeket városok szerint böngészhetnek
- a munkatársak egy grafikus felületű alkalmazásban szerkeszthetik az apartmanok adatait, képeit



# Objektumorientált tervezés

## 3. esettanulmány

- *Használati esetek:*



# Objektumorientált tervezés

## 3. esettanulmány

---

- *Tervezés* (komponensek, telepítés):
  - a rendszerben található egy webes, valamint egy adminisztrációs kliens, amelyet külön alkalmazások valósítanak meg
  - a webes kliens egy weblap, amelyet egy webszerverrel futtatunk, és ASP.NET keretrendszer segítségével valósítjuk meg
  - az adminisztrációs kliens egy asztali alkalmazás, amelyet .NET keretrendszerben valósítunk meg, ezért a .NET virtuális gépe (CLR) futtatja

# Objektumorientált tervezés

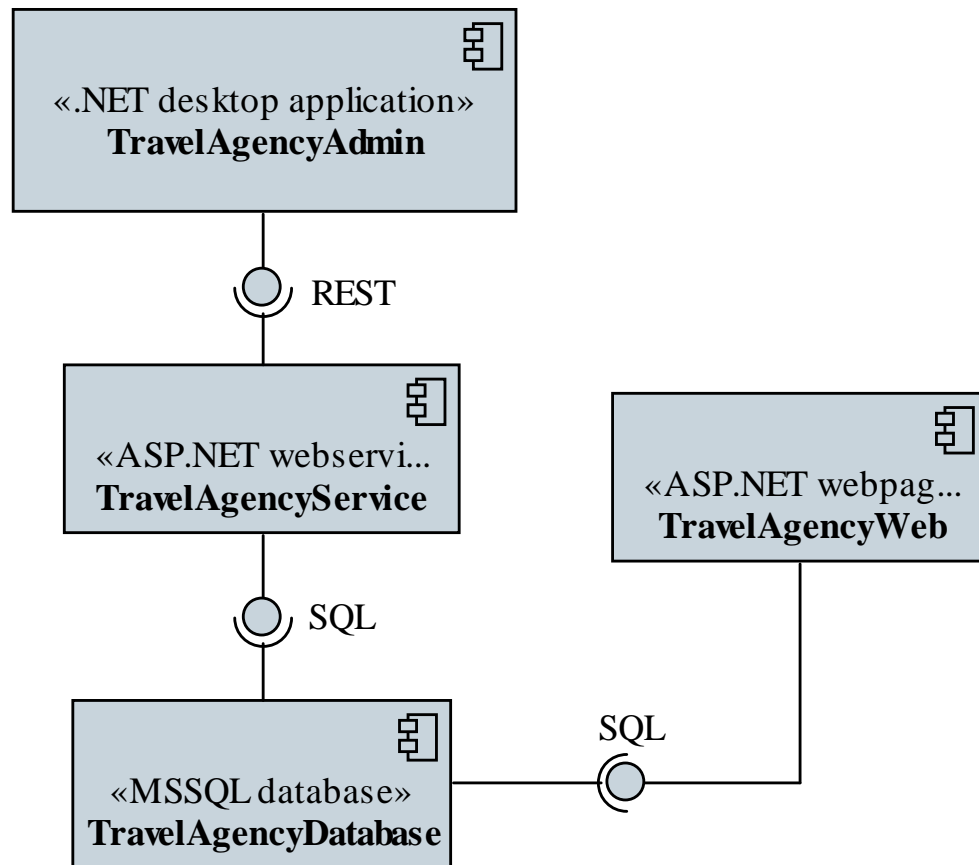
## 3. esettanulmány

- *Tervezés* (komponensek, telepítés):
  - a két alkalmazás közös adatokat használ, amelyeket relációs adatbázisban tárolunk, ehhez MSSQL-t használunk
  - a weblap és az adatbázis egy közös szerveren helyezkedik el, így a weblap közvetlenül hozzáfér az adatbázishoz
  - az asztali alkalmazás más számítógépen fog futni, ezért biztonsági okokból nem férhet hozzá közvetlenül az adatbázishoz, a hozzáféréshez közbeiktatunk egy webszolgáltatást
  - A webszolgáltatást egy webszerverrel futtatjuk, és ASP.NET/WebAPI keretrendszer segítségével valósítjuk meg

# Objektumorientált tervezés

## 3. esettanulmány

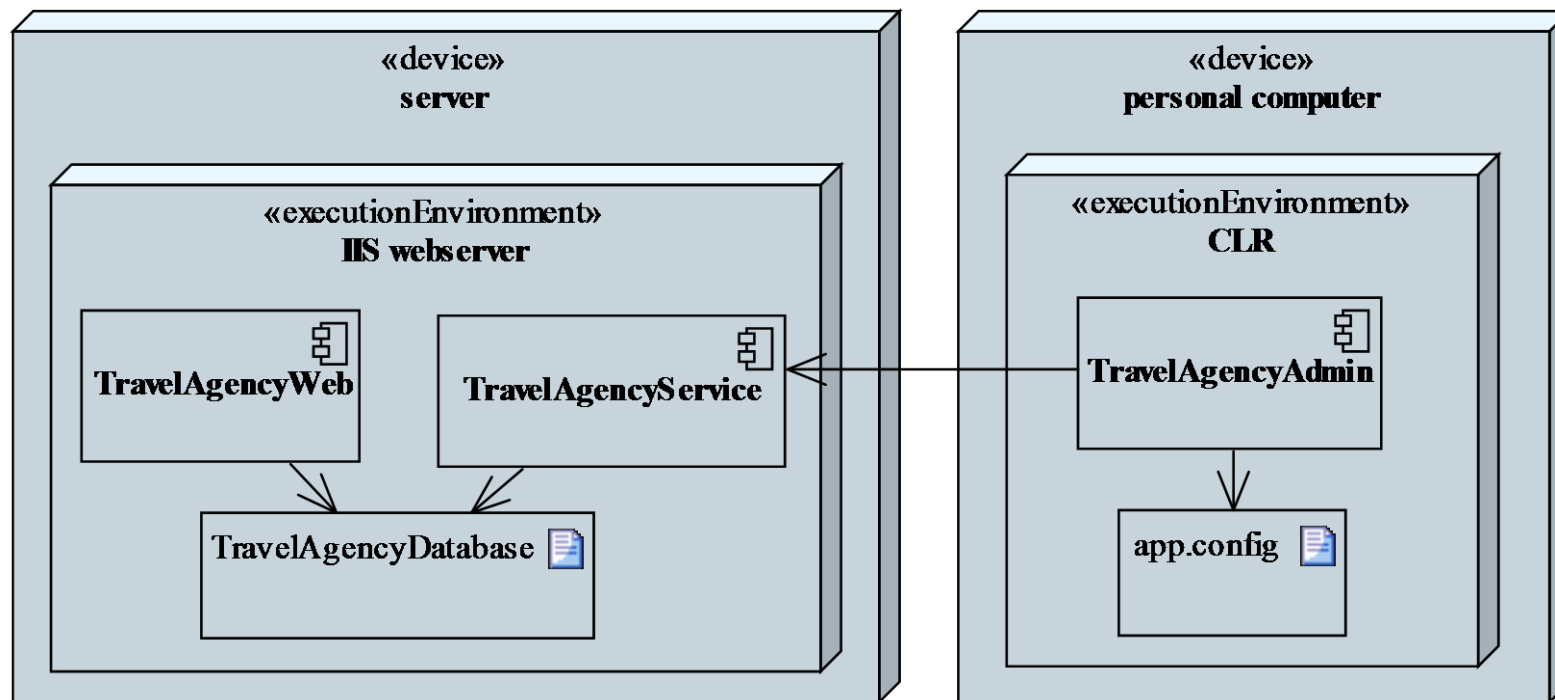
- *Tervezés* (komponensek):



# Objektumorientált tervezés

## 3. esettanulmány

- *Tervezés (telepítés):*



# Objektumorientált tervezés

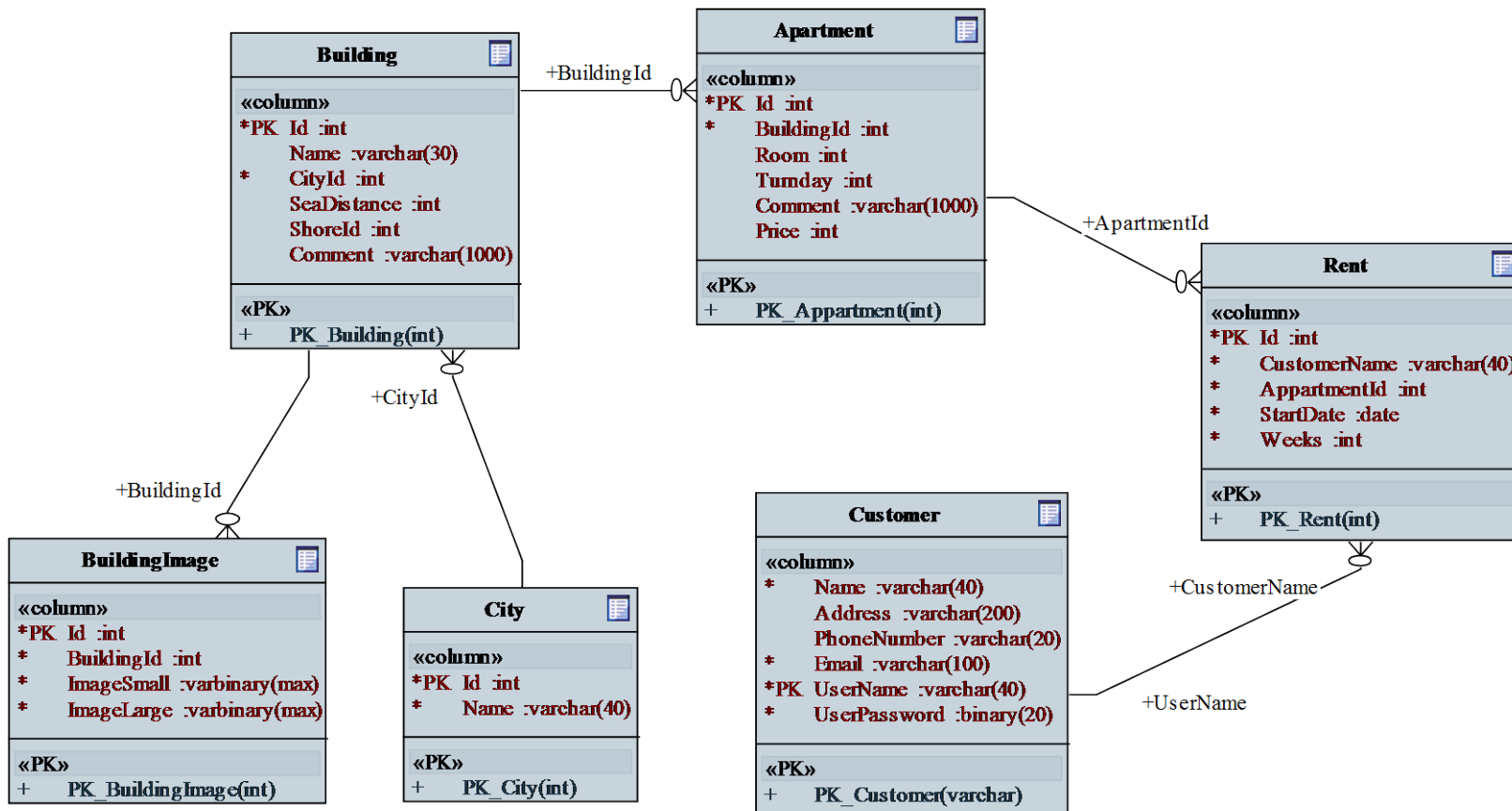
## 3. esettanulmány

- *Tervezés* (adatformátum):
  - az adatbázisban a következő séma szerint tároljuk az adatokat:
    - *városok* (*city*): azonosító, városnév;
    - *épületek* (*building*): azonosító, név, város azonosító, utca, tengerpart távolság, tengerpart-típus (számként), jellemzők (binárisan összeillesztve), megjegyzés;
    - *apartmanok* (*apartment*): azonosító, épület azonosító, szám, ágyak száma, pótágyak száma, felújítás alatt van-e;
    - *ügyfelek* (*customer*): azonosító, név;
    - ...

# Objektumorientált tervezés

## 3. esettanulmány

- *Tervezés (adatformátum):*



# Objektumorientált tervezés

## A rendszerterv

- A tervezés eredménye a *szoftver rendszerterve* (*software design description, SDD*), amely tartalmazza:
  - a program statikus szerkezetét, azaz a programegységek feladatát, részletes leírását és a köztük lévő relációkat
  - a program dinamikus szerkezetét, azaz a program eseményeinek kiváltódását és hatásait, a programegységek állapotainak változását, az üzenetküldések megvalósítását
  - a tárolt, kezelt, és eredményül adott adatok formáját, leírását
  - a programok belső és külső interfészeinek leírását
  - ajánlásokat az implementáció számára (stratégia, függőségek, programozási nyelv, tesztelési módszerek)



# Objektumorientált tervezés

## A rendszerterv

- A rendszerterv felépítése:
  1. előszó (célközönség, dokumentum-történet)
  2. bevezetés (szoftver célja, helye, szükségessége, előnyei, fejlesztési módszertan)
  3. fogalomtár (technikai áttekintés)
  4. rendszer architektúra (magas szintű áttekintés, UML csomag-, komponens-, állapotdiagram)
    - architektúrális minták
    - funkcionális megfeleltetés
  5. adat tervezés (adatformátumok leírása)

# Objektumorientált tervezés

## A rendszerterv

- A rendszerterv felépítése:
  6. rendszer tervezés (alacsony szintű áttekintés)
    - statikus terv (UML osztály-, objektumdiagram)
    - dinamikus terv (UML állapot-, szekvencia- és aktivációs diagram)
    - interfész leírás
    - felhasznált algoritmusok és minták
  7. felhasználói felület (áttekintés, felületi terv)
  8. implementációs ajánlások
  9. függelék (pl. adatbázis terv, becsült hardver szükségletek)
  10. tárgymutató