

**Eötvös Loránd Tudományegyetem
Informatikai Kar**

Szoftvertchnológia

6. előadás

Objektumorientált tervezés: állapotkezelés

Giachetta Roberto

groberto@inf.elte.hu

<http://people.inf.elte.hu/groberto>



Objektumorientált tervezés: állapotkezelés

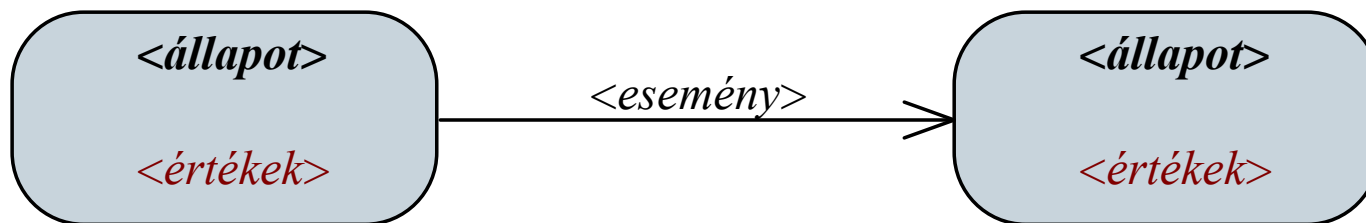
Objektumok állapota

- Az objektumok *állapottal* (state) rendelkeznek, ahol az állapot adatértékeinek összessége
 - két objektum állapota ugyanaz, ha értékeik megegyeznek (ettől függetlenül az objektumok különbözőek)
 - az állapot valamilyen *esemény* (műveletvégzés, kommunikáció) hatására változhat meg
 - amennyiben egy objektum csak egy állapottal rendelkezik, akkor az objektum *változtathatalan* (*immutable*)
- Az objektumok *életciklussal* rendelkeznek, létrejönnek (a konstruktorral), működést hajtanak végre (további műveletekkel), majd megsemmisülnek (a destruktorkal)

Objektumorientált tervezés: állapotkezelés

Állapotok modellezése

- Amennyiben egy objektumorientált szoftver működését szeretnénk megvizsgálni, fontos szerepet tölt be az objektumok lehetséges állapotainak modellezése
 - mivel a lehetséges állapotok a viselkedési mintától függenek, ezért az osztályok alapján állapítjuk meg
- Az objektumok állapotait, és a köztük lévő állapotátmeneteket az *állapotdiagram*, vagy *állapotautomata (state machine)* segítségével tudjuk modellezni



Objektumorientált tervezés: állapotkezelés

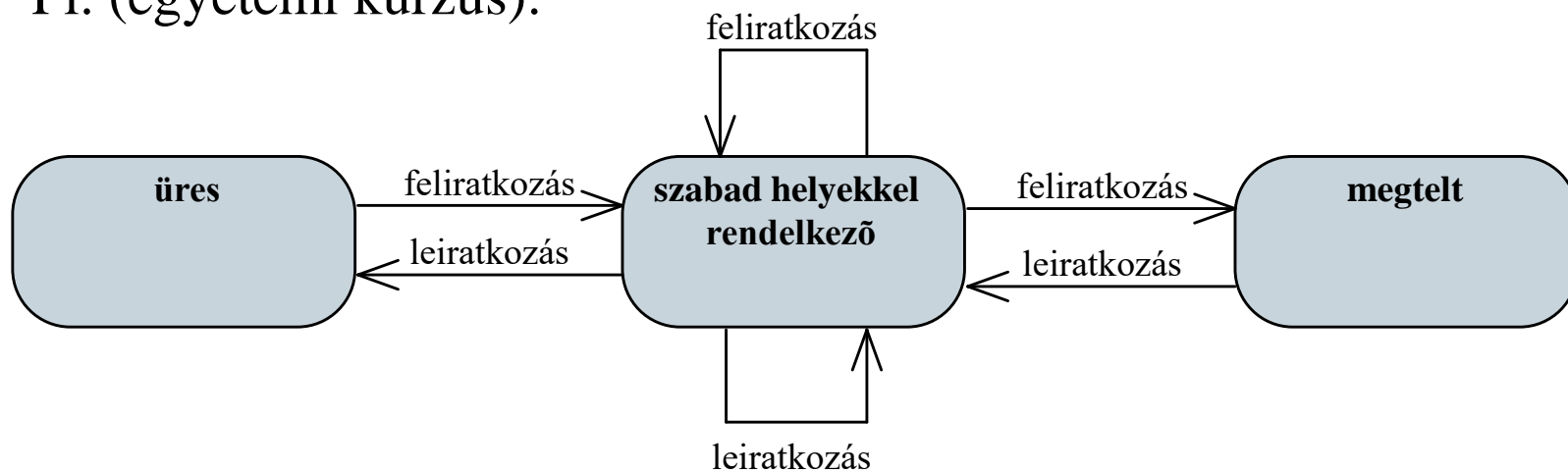
Állapotok modellezése

- *Állapot (state)*: az objektum értékeinek (mezőinek) összessége, amely befolyásolja az objektum felhasználhatóságát, az objektummal végezhető tevékenységeket
 - rendelkezik egy (egyedi) *elnevezéssel*
 - megadhatjuk benne a konkrét értékeket, vagy használhatjuk csupán az elnevezést
 - általában nem egy konkrét érték kombinációt, hanem érték kombinációk lehetséges halmazát ábrázolja, amelyekre egy adott feltétel, az *állapotinvariáns* fennáll
 - az állapot addig marad fent, amíg az adatok értékei kielégítik az invariánst

Objektumorientált tervezés: állapotkezelés

Állapotok modellezése

- *Esemény (event, trigger)*: az objektummal végzett tevékenység, kommunikáció (metódushívás, eseménykiváltás)
 - lehet *elnevezése, paramétere, feltétele és hatása*
 - lehet reflexív, ebben az esetben nem okoz állapotváltást, és *belső eseménynek* nevezzük
- Pl. (egyetemi kurzus):



Objektumorientált tervezés: állapotkezelés

Állapotok modellezése

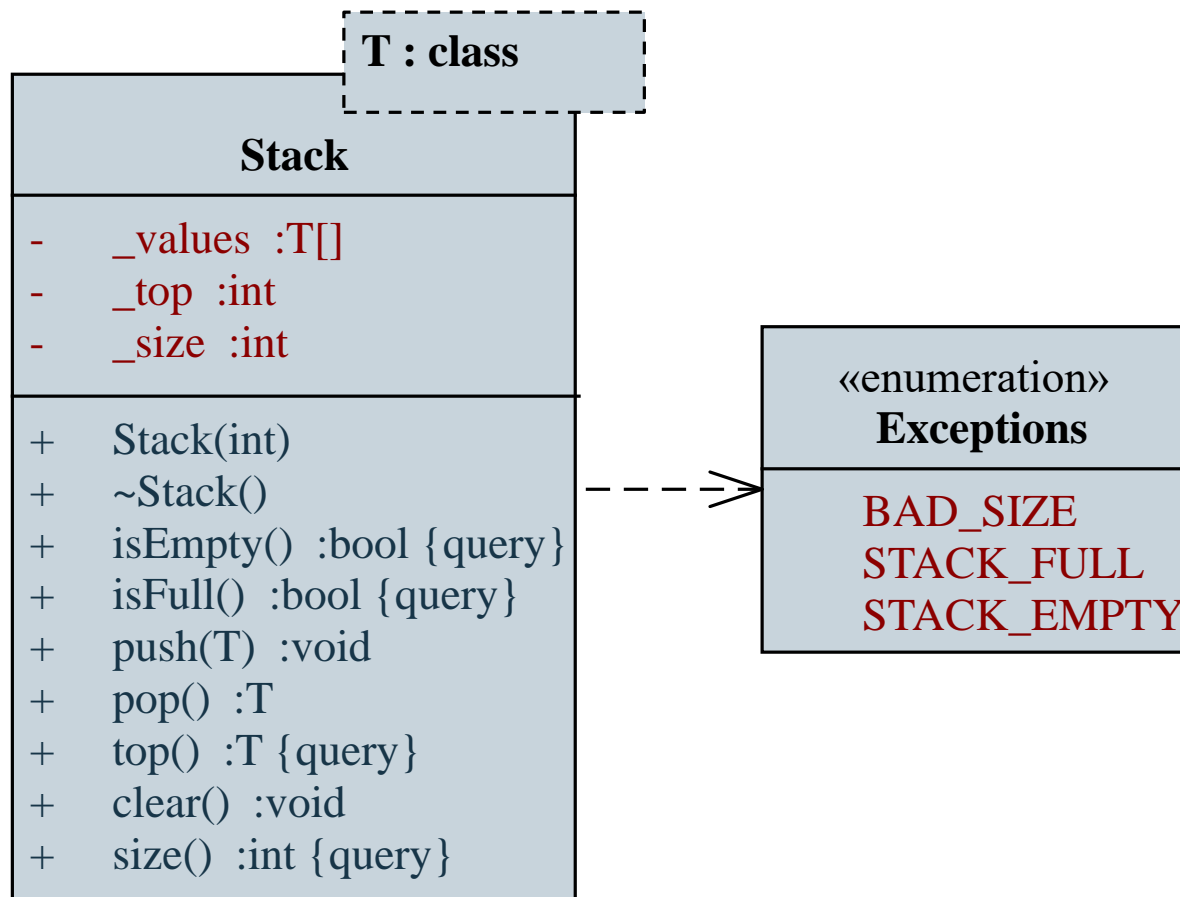
Feladat: Valósítsuk meg a verem (**Stack**) adatszerkezetet aritmetikai reprezentáció mellett. Lehessen elemet behelyezni (**push**), kivenni (**pop**), kitörölni a teljes vermet (**clear**), lekérdezni a tetőelemet (**top**), üres-e (**isEmpty**), illetve tele van-e a verem (**isFull**), valamint mi az aktuális mérete (**size**).

- a verem működését a tömbben lévő elemek száma befolyásolja
- ha nincs elem a tömbben, nem lehet kivenni, lekérdezni
- ha a tömbben lévő elemek száma eléri a maximumot, nem lehet behelyezni
- ennek megfelelően három különböző állapotot különíthetünk el (**empty**, **normal**, **full**)

Objektumorientált tervezés: állapotkezelés

Állapotok modellezése

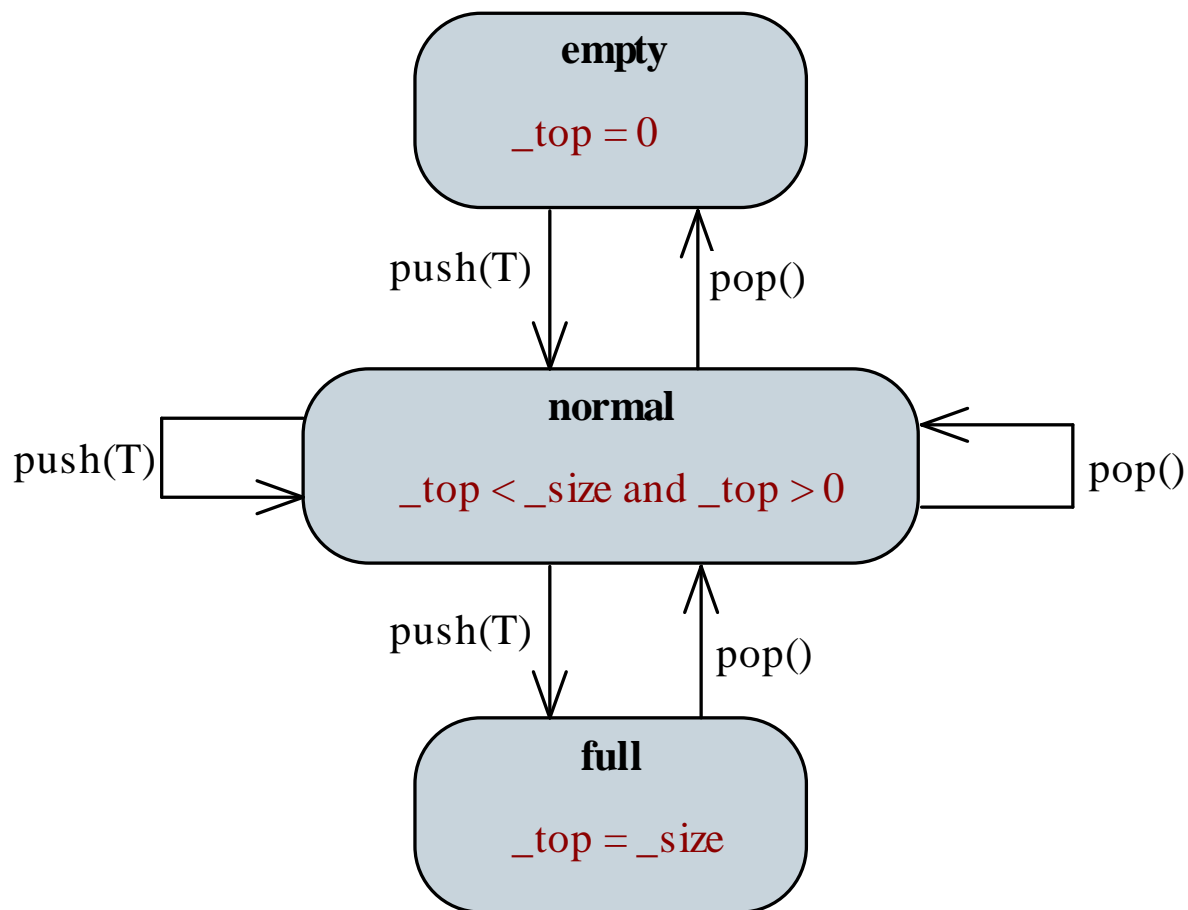
Szerkezeti tervezés:



Objektumorientált tervezés: állapotkezelés

Példa

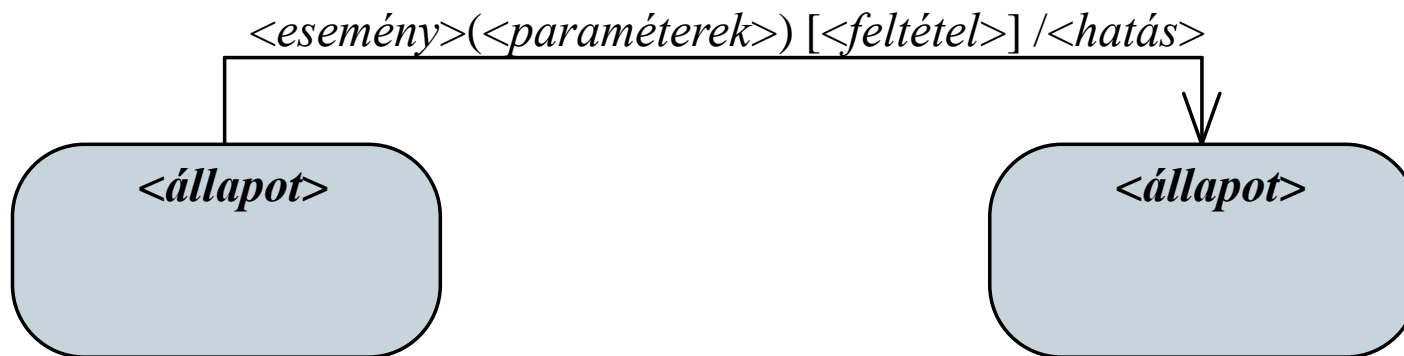
Dinamikus tervezés (állapotkezelés):



Objektumorientált tervezés: állapotkezelés

Feltételek és hatás

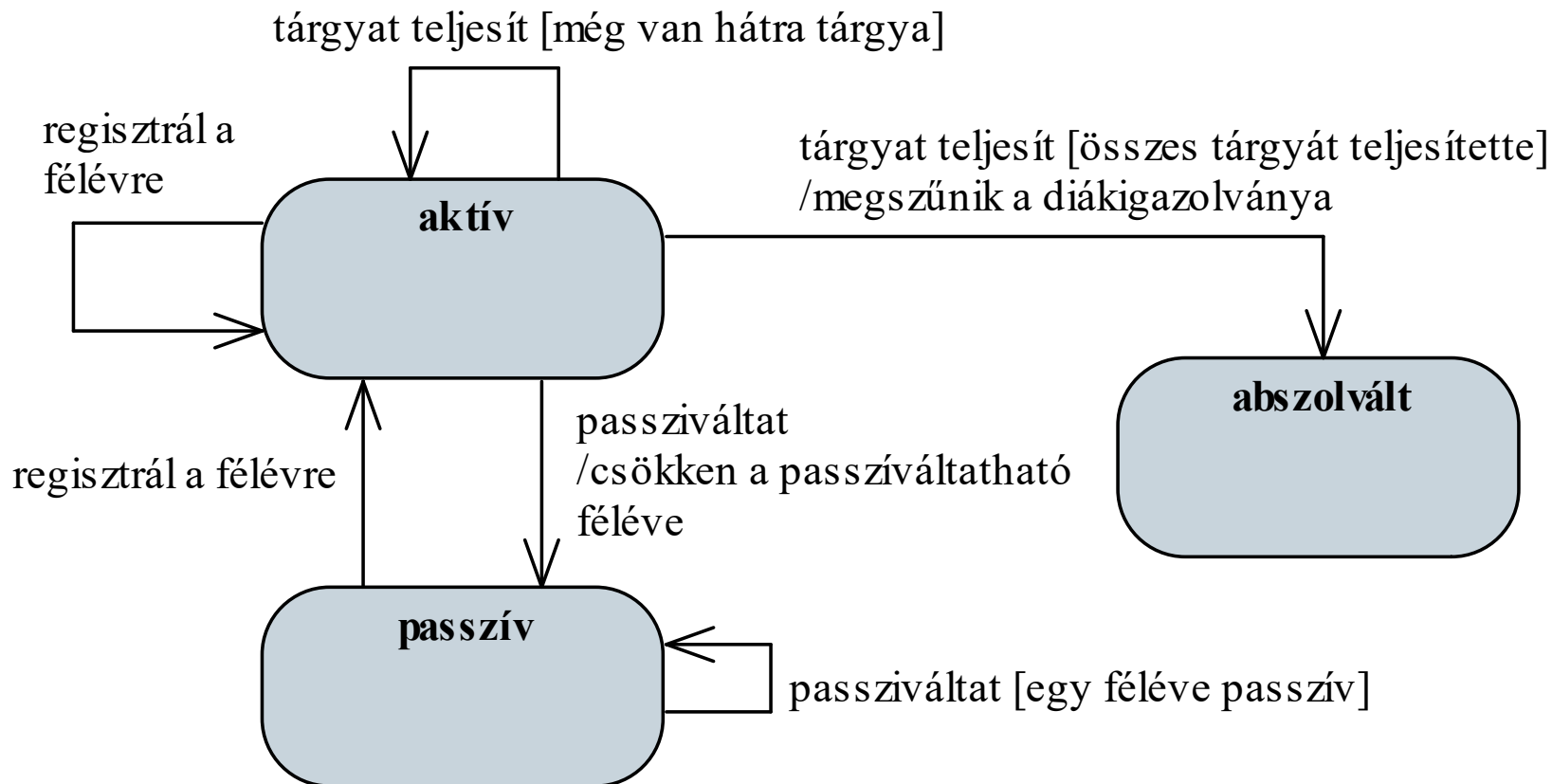
- Az esemény rendelkezhet:
 - *paraméterekkel*, amelyek az eseményre vonatkoznak
 - *feltétellel* (*guard*), amelynek a teljesülése szükséges az átmenet bekövetkeztével
 - *hatással* (*effect*), amely egy, az állapotátmenet hatására végrehajtott tevékenység (*akció*)



Objektumorientált tervezés: állapotkezelés

Feltételek és hatás

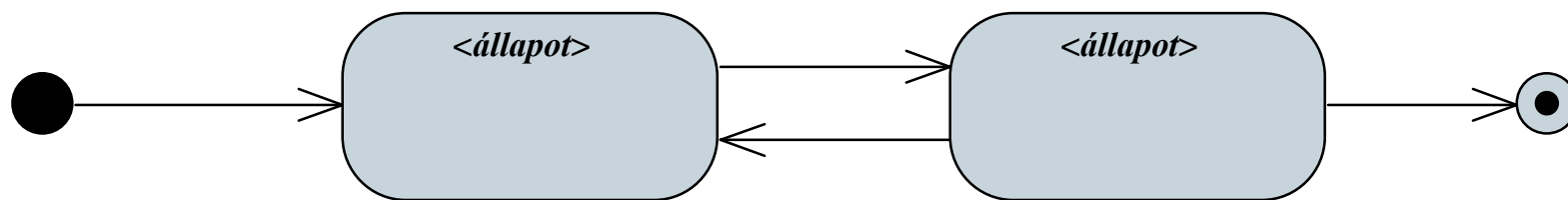
- Pl. (egyetemi hallgató):



Objektumorientált tervezés: állapotkezelés

Kezdő és végállapot

- Az objektumok létrehozásukkor egy *kezdeti állapotból* (*initial state*) indulnak, majd életciklusuk végén egy *végállapottal* (*final state*) terminálnak

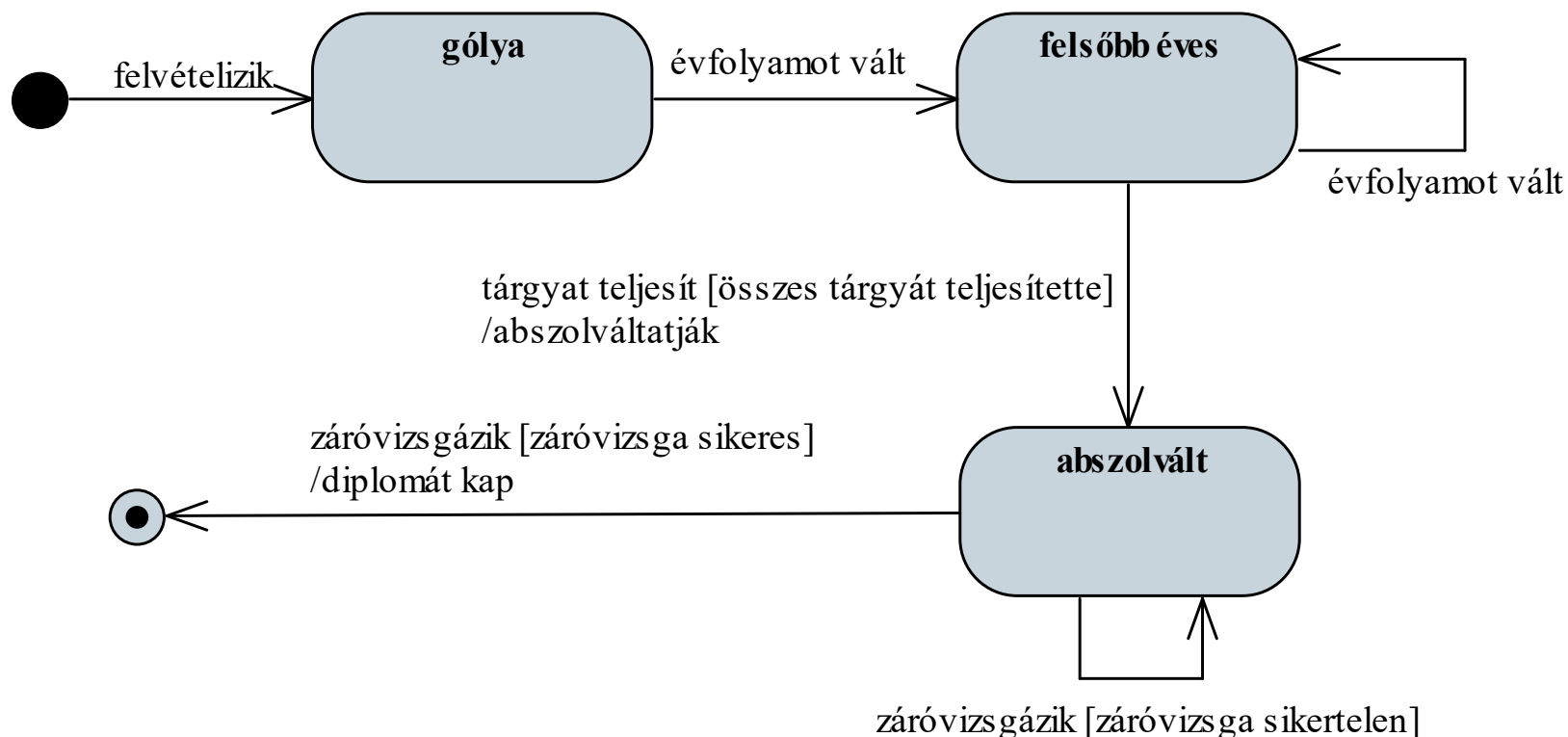


- a kezdőállapotnak és végállapotnak általában nem adunk nevet
- kezdőállapotba és végállapotból nem vezet átmenet, a kezdőállapotból történő átmenetnek nincs feltétele

Objektumorientált tervezés: állapotkezelés

Kezdő és végállapot

- Pl. (egyetemi hallgató):



Objektumorientált tervezés: állapotkezelés

Kezdő és végállapot

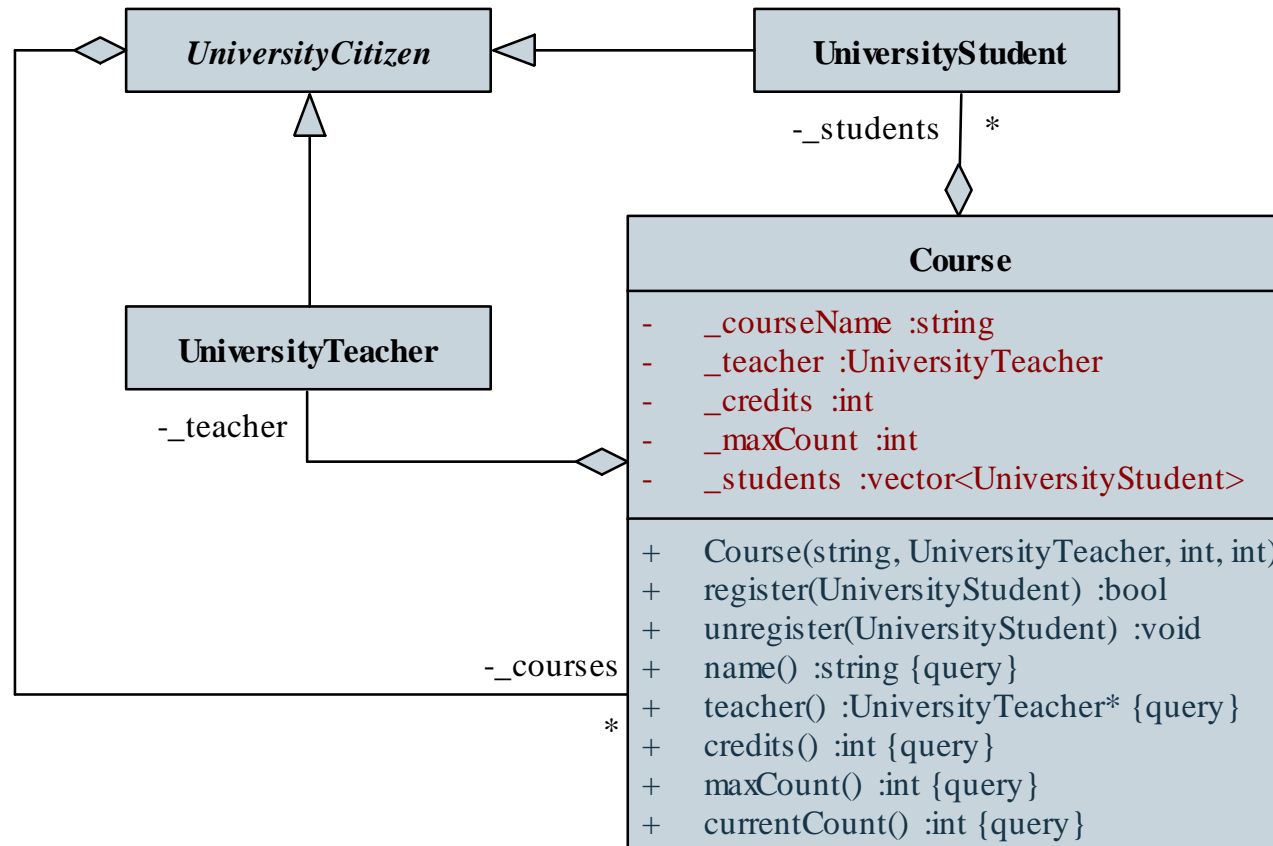
Feladat: Készítsünk egy programot, amelyben egyetemi oktatók, hallgatók és kurzusok adatait tudjuk tárolni.

- a kurzus (**Course**) rendelkezik névvel, oktatóval, hallgatókkal, kreditszámmal és maximális létszámmal
- a hallgató felveheti a kurzust (**register**), amennyiben még van szabad hely, és még nem jelentkezett rá (ekkor a kurzus megjelenik a hallgatónál is), és lejelentkezhet róla (**unregister**), amennyiben jelentkezett már rá (ekkor a kurzust a hallgatótól is elvesszük)
- kezdetben a kurzus üres (nincs jelentkezett hallgató), és csak üres kurzus törölhető

Objektumorientált tervezés: állapotkezelés

Kezdő és végállapot

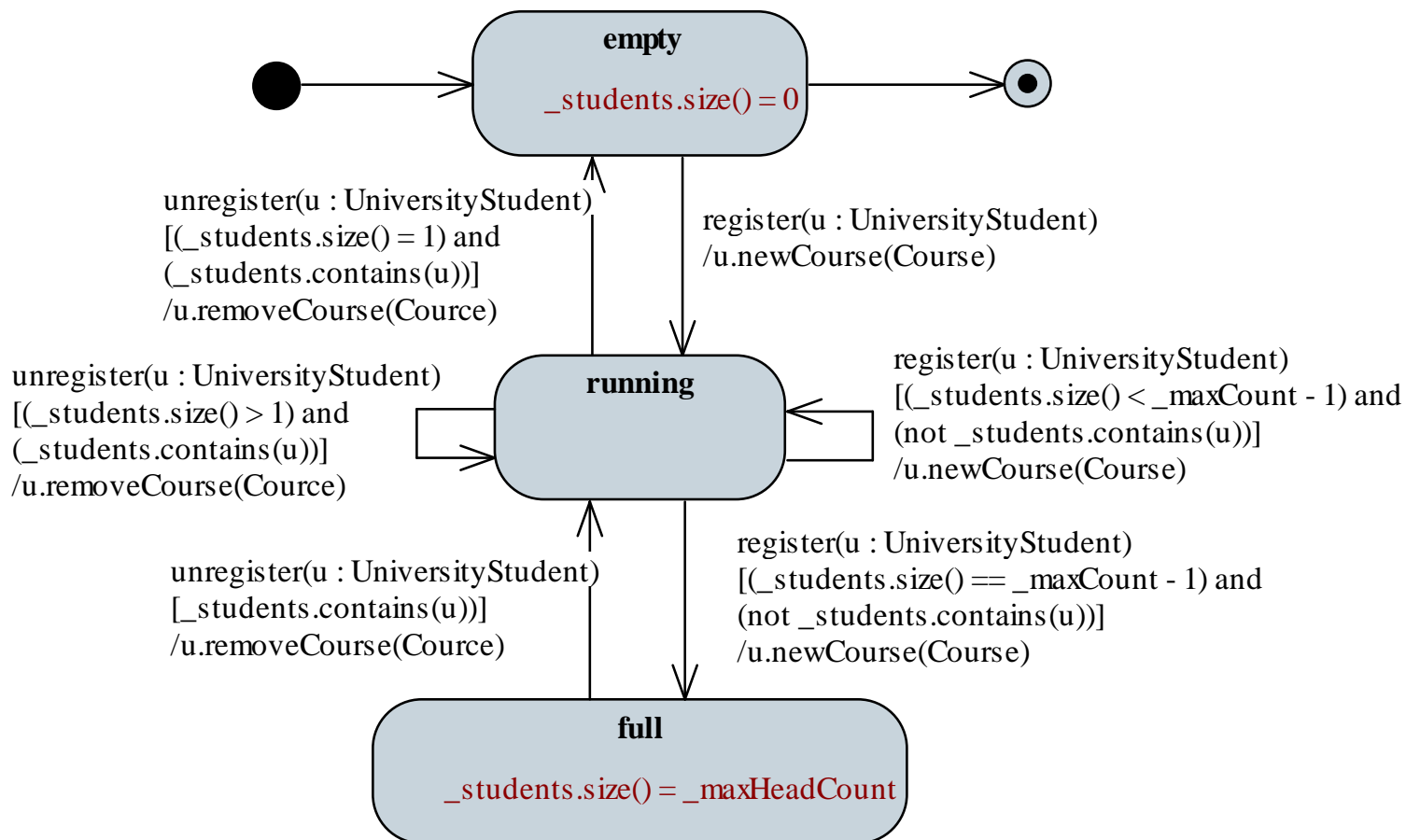
Szerkezeti tervezés:



Objektumorientált tervezés: állapotkezelés

Kezdő és végállapot

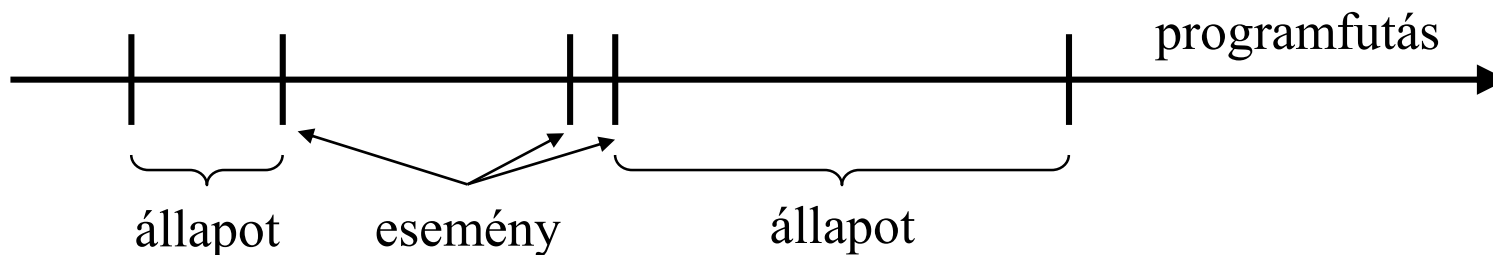
Dinamikus tervezés (kurzus állapotai):



Objektumorientált tervezés: állapotkezelés

Tevékenység állapotok

- Az állapothoz általában a működésben egy időintervallum, az eseményhez egy időpont tartozik

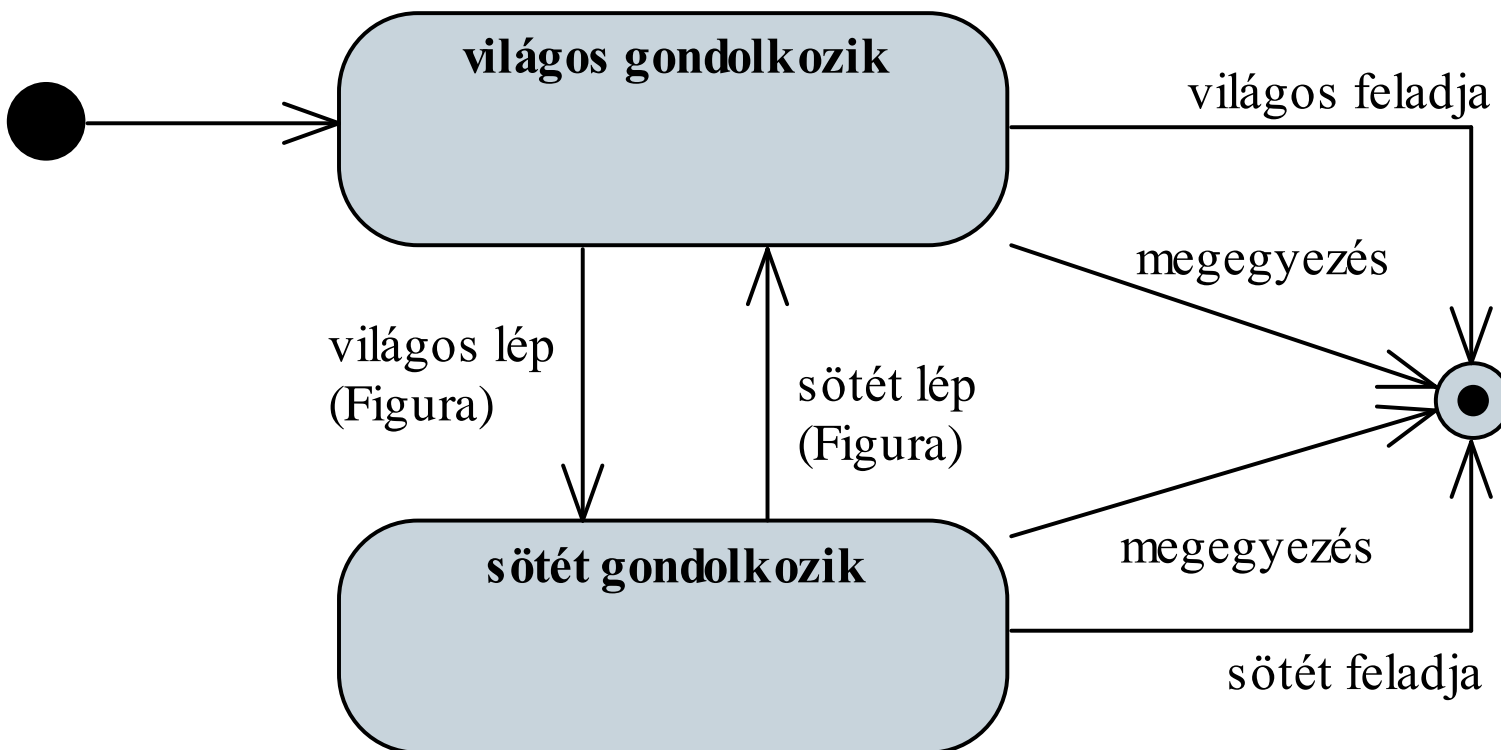


- Ugyanakkor egy adott tevékenység időben is elhúzódhat, így magát a tevékenységet is ábrázolhatjuk állapotként
 - a tevékenység állapotba vezető, és onnan kivezető esemény sokszor csak a tevékenység kezdetét és végét jelenti, ezért nem kap külön elnevezést

Objektumorientált tervezés: állapotkezelés

Tevékenység állapotok

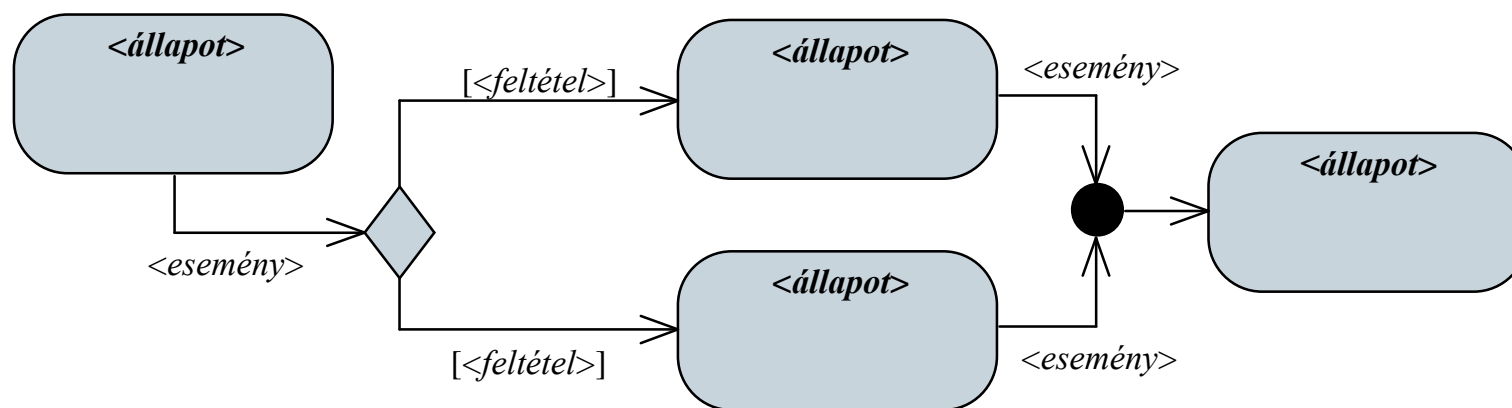
- Pl. (sakkjátzsma):



Objektumorientált tervezés: állapotkezelés

Elágazások

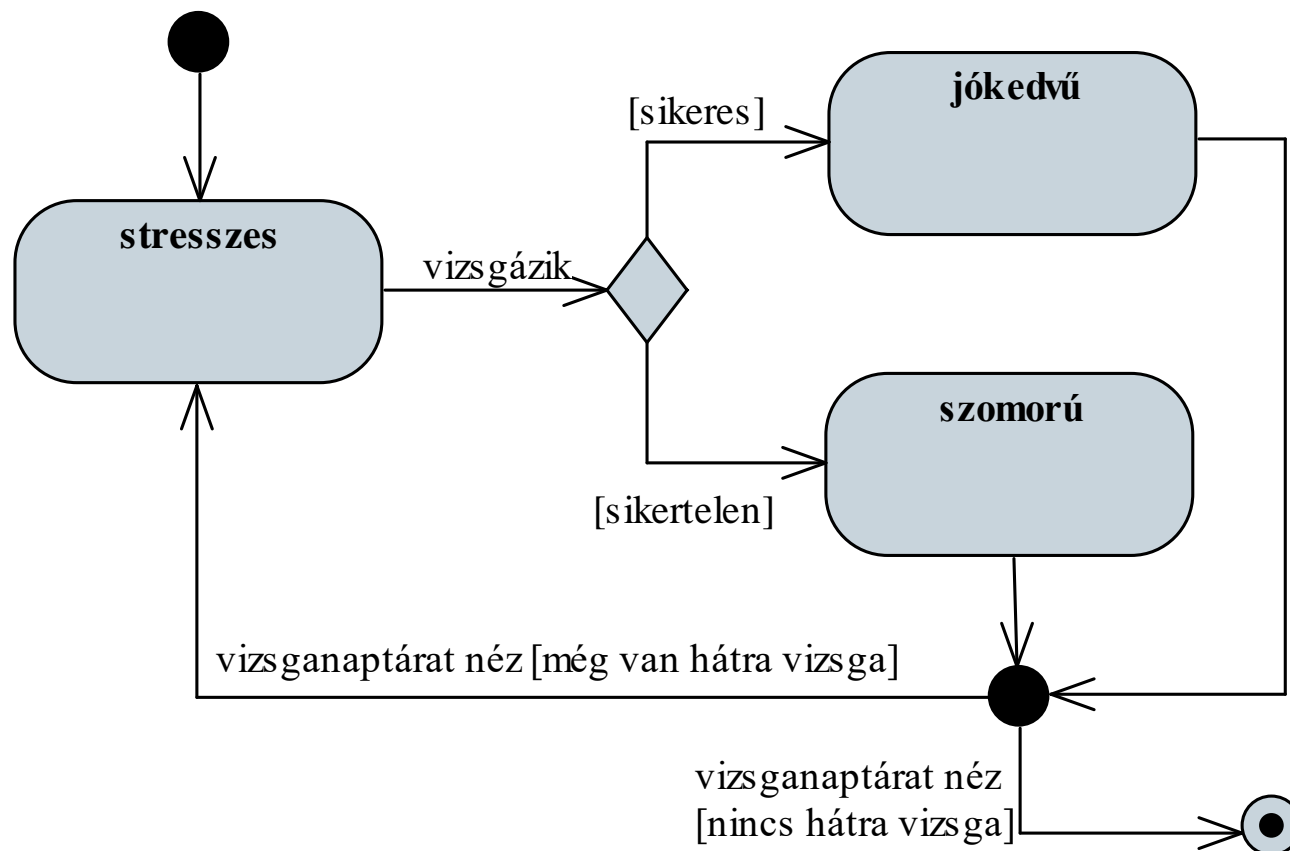
- Az állapotváltások során könnyen előfordulhatnak elágazások, ciklusok, amelyeket modellezhetünk
 - feltételhez kötött eseménnyel
 - választással (*choice*), ahol csak a feltételeket adjuk meg
- Az elágazás végeztével újra egy pontban egyesül az állapot, amely ábrázolható *csomóponttal (junction)*



Objektumorientált tervezés: állapotkezelés

Általánosított állapotok

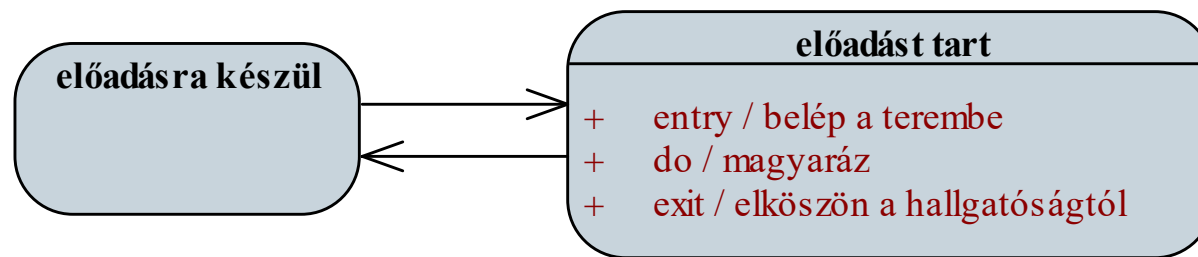
- Pl. (egyetemi hallgató vizsgaidőszaka):



Objektumorientált tervezés: állapotkezelés

Állapotok leírása

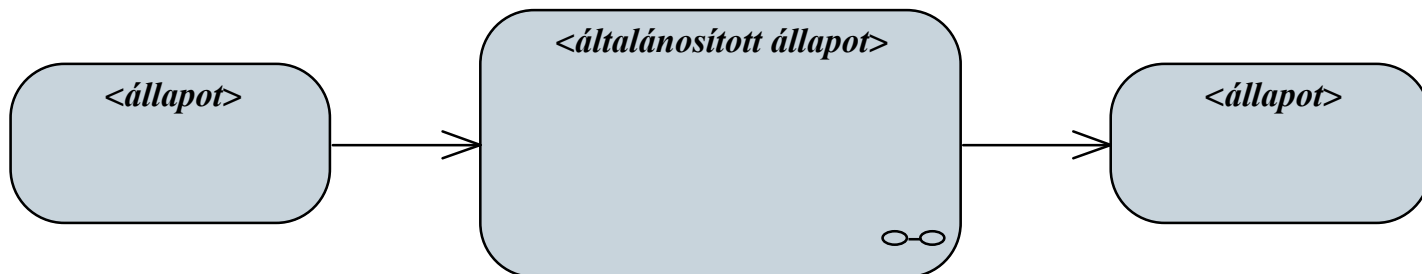
- Az állapoton belül is megfogalmazhatjuk a tevékenységeket, ehhez három fázist vehetünk fel
 - a *belépési (entry)* fázis tartalmazza az állapotba való belépéskor végrehajtandó tevékenység
 - a *végrehajtó (do)* fázis az állapotban végrehajtandó tevékenység
 - a *kilépési (exit)* fázis az állapotból kivezető tevékenység
- Pl. (egyetemi oktató):



Objektumorientált tervezés: állapotkezelés

Általánosított állapotok

- Lehetőségünk az állapotok *általánosítására* (*generalization*), amely egy általános állapottal ábrázolunk több lehetséges állapotot, és a köztük lévő átmenetet

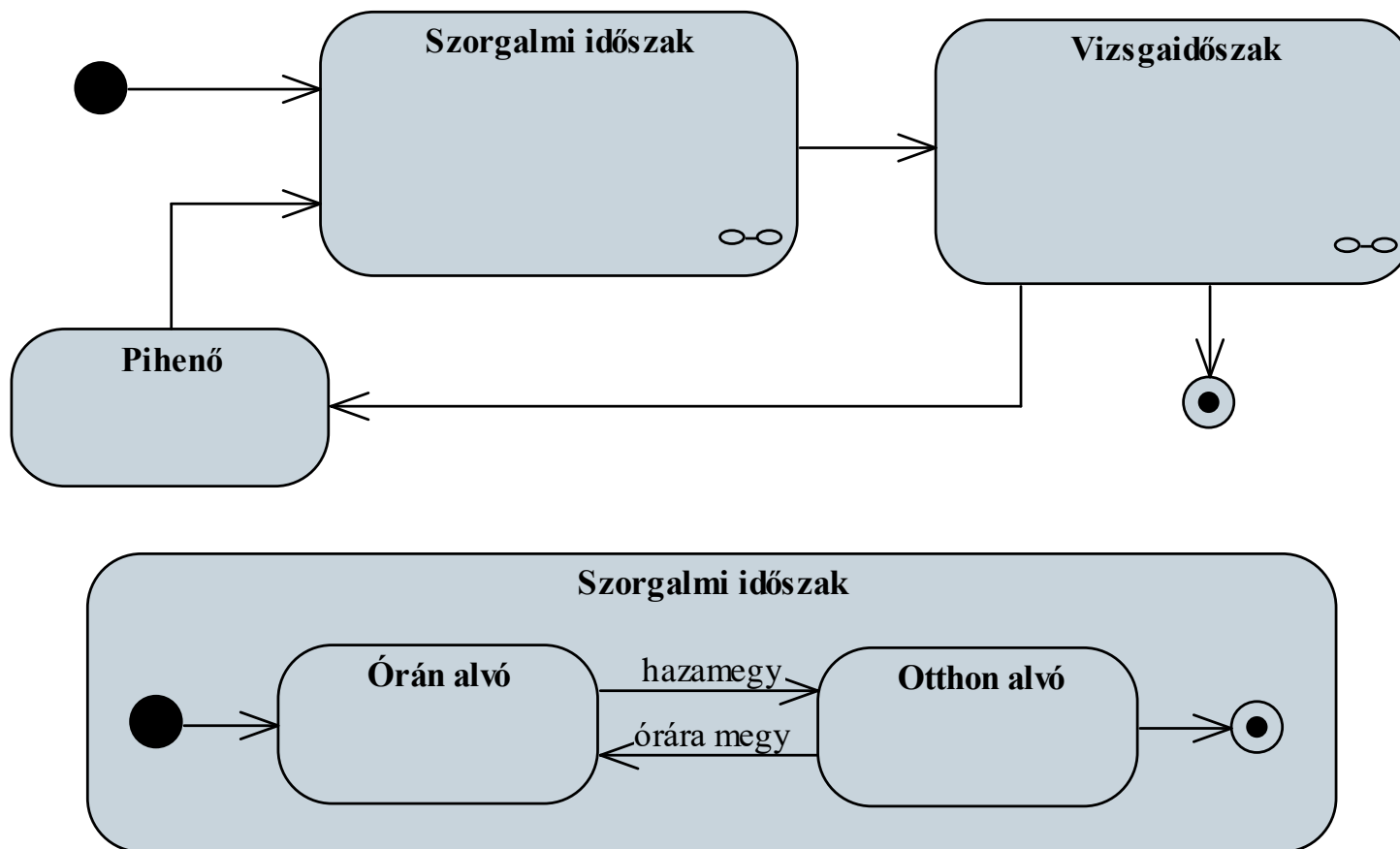


- ekkor az általános állapot is egy állapotautomata lesz, amelynek tartalmát kifejthetjük (a későbbiekben)
- az általánosított állapot belépési pontja lesz az állapotautomata kezdőállapota, kilépési pontja pedig a végállapota

Objektumorientált tervezés: állapotkezelés

Általánosított állapotok

- Pl. (egyetemi hallgató):



Esettanulmányok

Tic-Tac-Toe játék

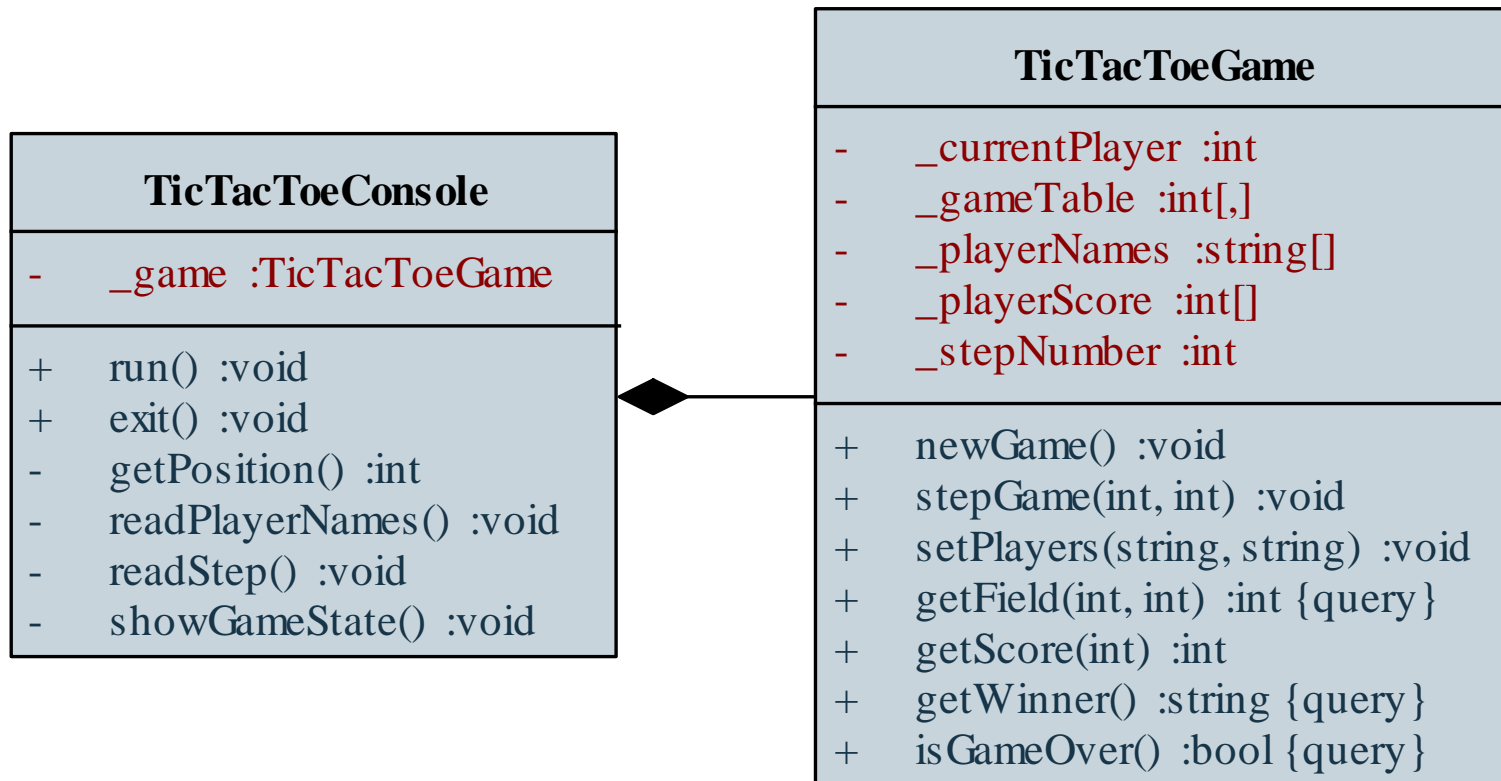
Feladat: Készítsünk egy Tic-Tac-Toe programot, amelyben két játékos küzdhet egymás ellen.

- a programban jelenjen meg egy játéktábla, amelyen végig követjük a játék állását (a két játékost az ,X' és ,0' jelekkel ábrázoljuk)
- legyen lehetőség a játékosok neveinek megadására, új játék indítására, valamint játékban történő lépésre (felváltva)
- a program kövesse végig, melyik játékos hány kört nyert
- program automatikusan jelezzen, ha vége egy játéknak, és jelenítse meg a játékosok pontszámait

Esettanulmányok

Tic-Tac-Toe játék

Szerkezeti tervezés:



Esettanulmányok

Marika néni kávézója

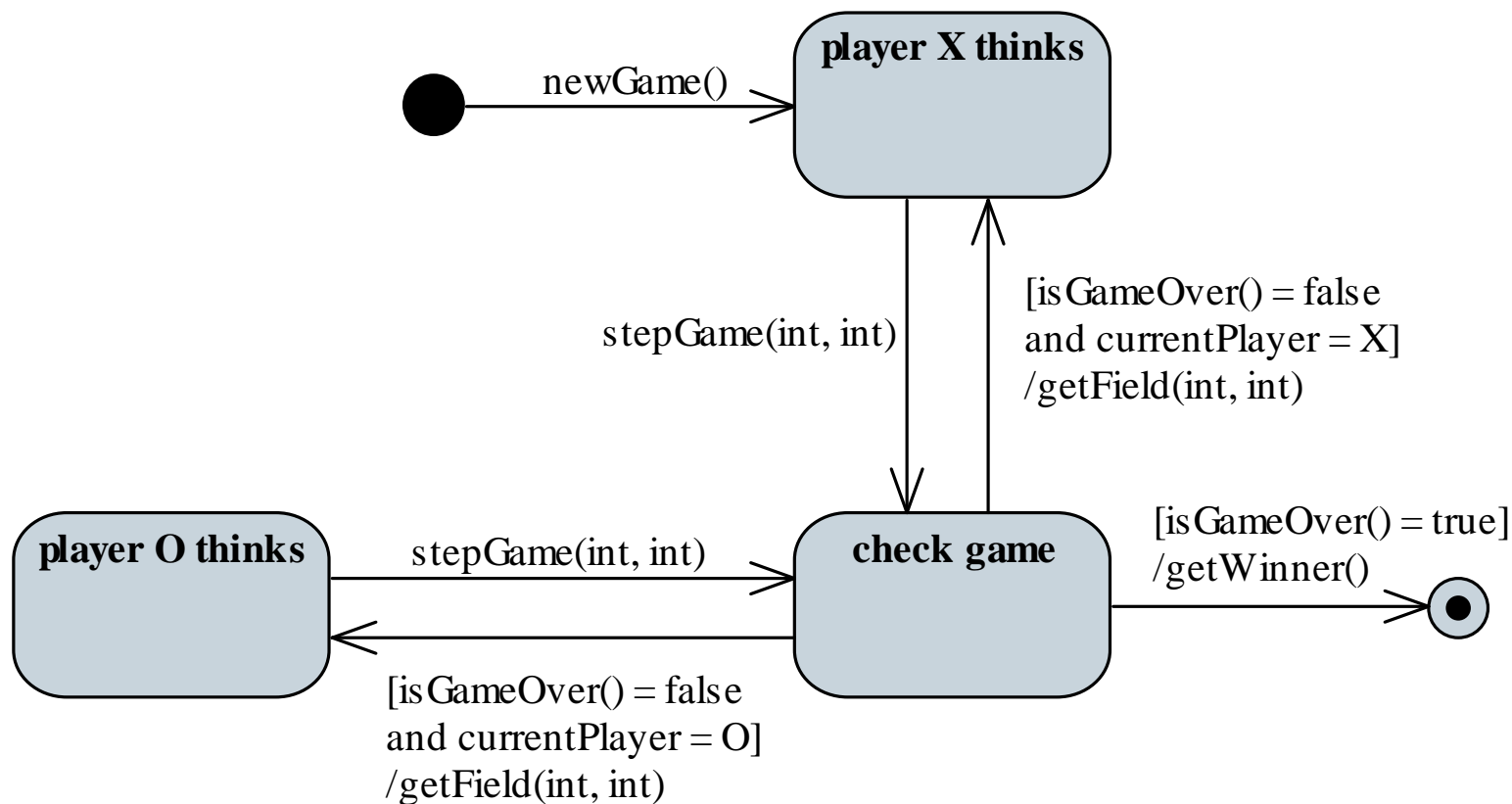
Dinamikus tervezés (állapotkezelés):

- A játék menetének állapotai:
 - új játék kezdését (**newGame**) követően az első játékos (X) gondolkodik, majd lép (**stepGame**)
 - lépést követően ellenőrizni kell a játékállást
 - amennyiben nincs vége a játéknak (**isGameOver**), megjelenítjük a játékállást (**getField**), majd a másik játékos következik
 - a másik játékos (O) gondolkodik, lép, majd ismét ellenőrizni kell a játékállást
 - amennyiben vége van a játéknak, lekérjük a győztes nevét (**getWinner**)

Objektumorientált tervezés: állapotkezelés

Kezdő és végállapot

Dinamikus tervezés (állapotkezelés):



Esettanulmányok

Marika néni kávézója

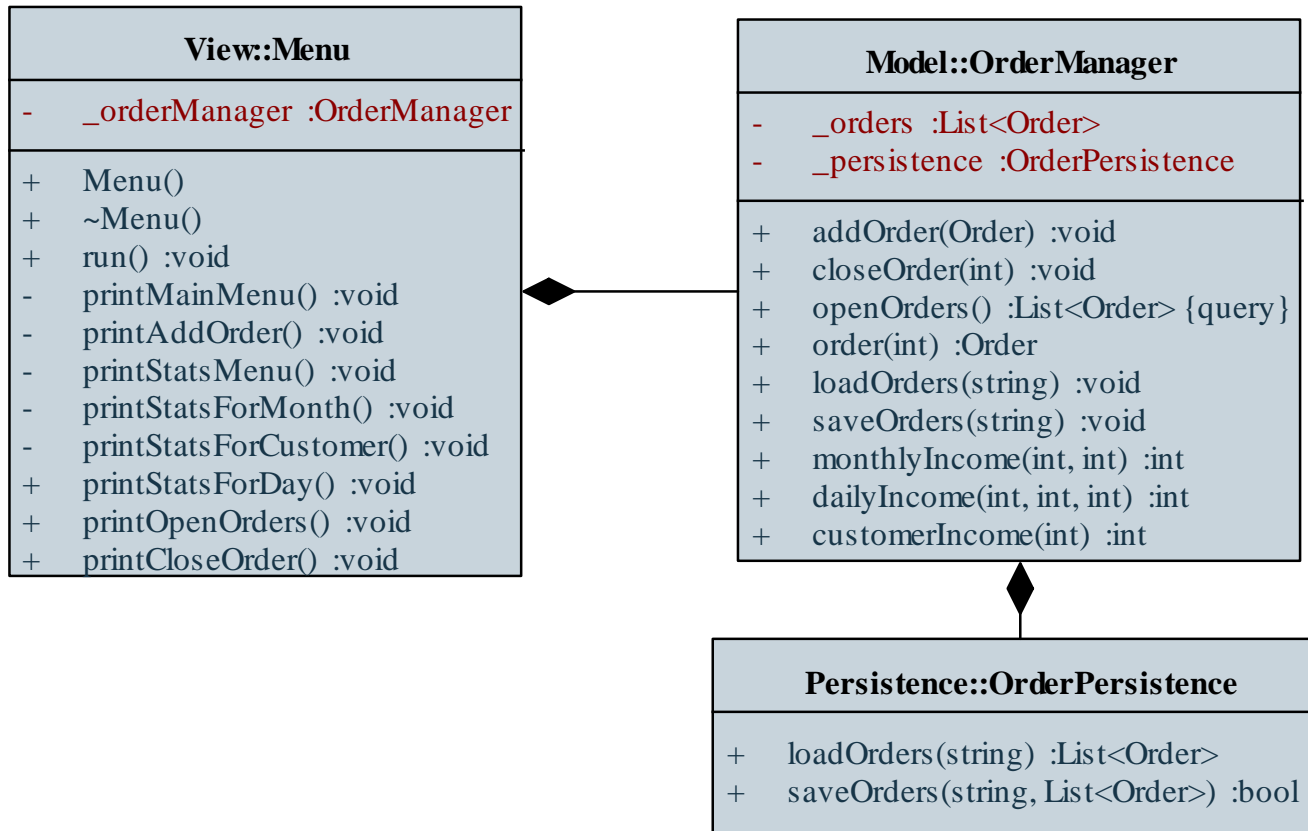
Feladat: Készítsük el Marika néni kávézójának eladási nyilvántartását végigkövető programot.

- 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
- a program kezelje a rendeléseket, amelyekben tetszőleges tételek szerepelhetnek, illetve a rendelés kapcsolódhat egy törzsvásárlóhoz
- biztosítsunk lehetőséget a függőben lévő rendeléseket lekérdezésére, valamint napi, havi és törzsvásárlói számra összesített nettó/bruttó fogyasztási statisztikák követésére

Esettanulmányok

Marika néni kávézója

Szerkezeti tervezés:



Esettanulmányok

Marika néni kávézója

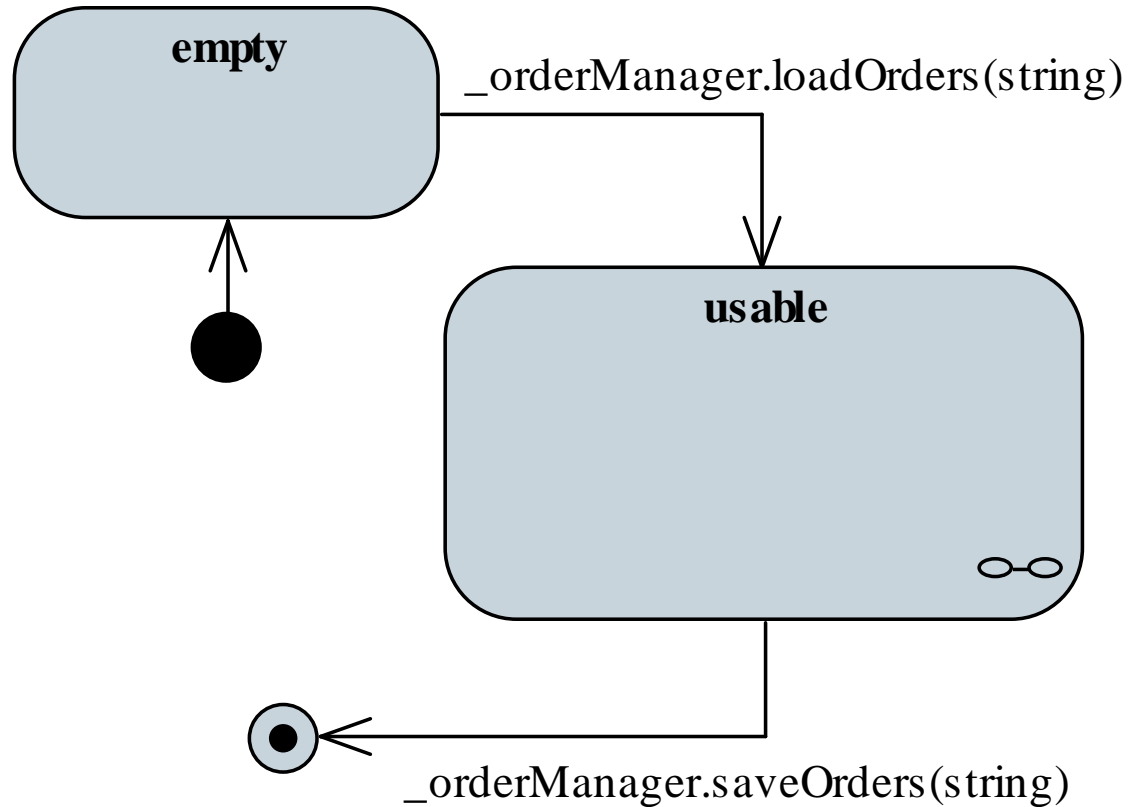
Dinamikus tervezés:

- A rendelés elemei változtathatatlan objektumok, ezért nincsenek állapotaik
- A rendelésen és a rendeléskezelőn végrehajtunk tevékenységeket, ugyanakkor állapotaik nincsenek kihatással a program futására, használatára
- A menü futtatása biztosítja a lényeges tevékenységeket a különböző menüpontok segítségével, itt megjelenhet a főmenü, egyes statisztikák, az új rendelés megadása, vagy a nyitott rendelések
 - a menü csak akkor lesz használható, ha az adatbetöltést elvégezzük, és természetesen terminálás előtt mentenünk is kell

Esettanulmányok

Marika néni kávézója

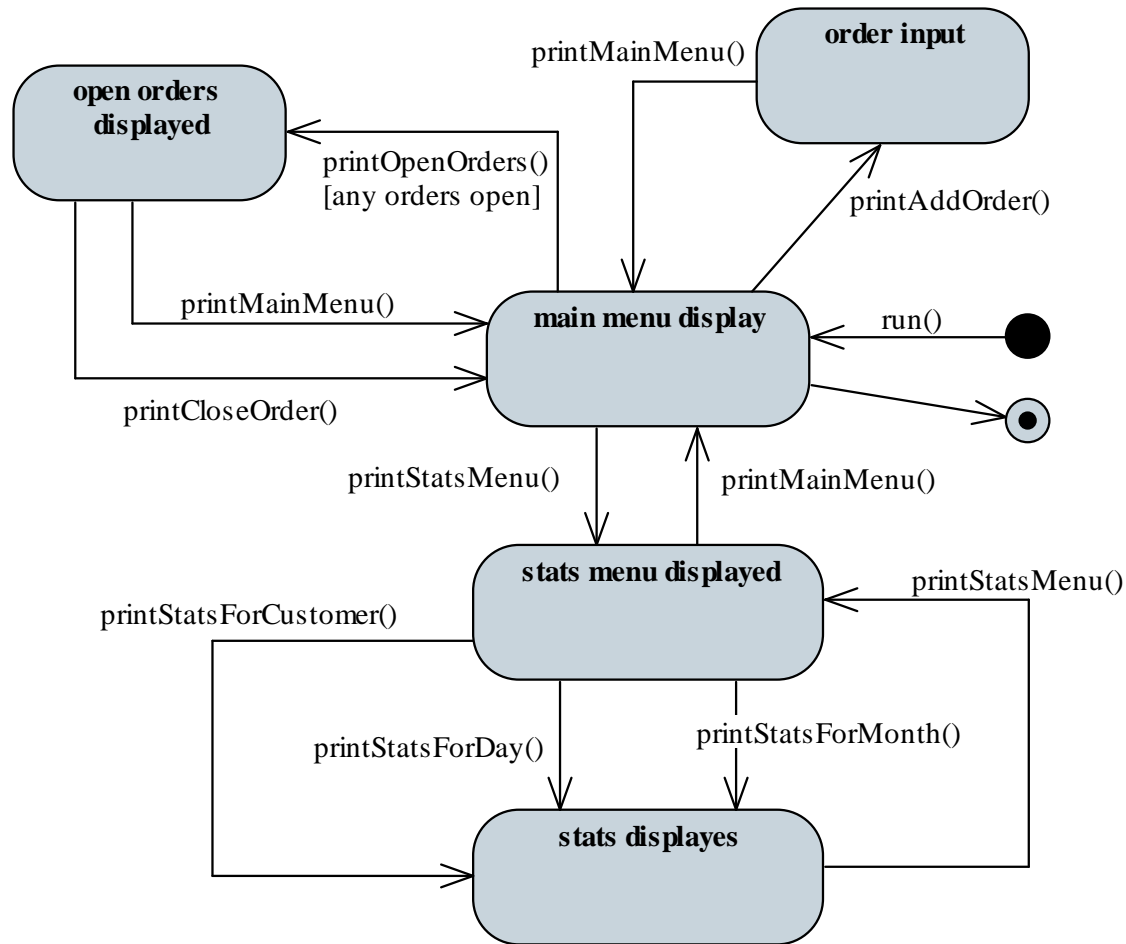
Dinamikus tervezés (Menu állapotai):



Esettanulmányok

Marika néni kávézója

Dinamikus tervezés (usable állapotai):



Esettanulmányok

Utazási ügynökség

Feladat: Készítsük el egy utazási ügynökség apartmanokkal foglalkozó rendszerét.

- az apartmanok épületekben találhatóak, amelyek városokban helyezkednek el
- az épületek különböző adatokkal (leírás, szolgáltatások, pontos hely, tengerpart távolság, ...), valamint képekkel rendelkeznek
- a vendégek számára biztosítsunk egy webes felületet, amelyen keresztül apartmanokat kereshetnek, foglalhatnak
- a munkatársak számára biztosítsunk egy alkalmazást, amelyben szerkeszthetik az apartmanok adatait, képeit, valamint kezelhetik a foglalásokat

Esettanulmányok

Utazási ügynökség

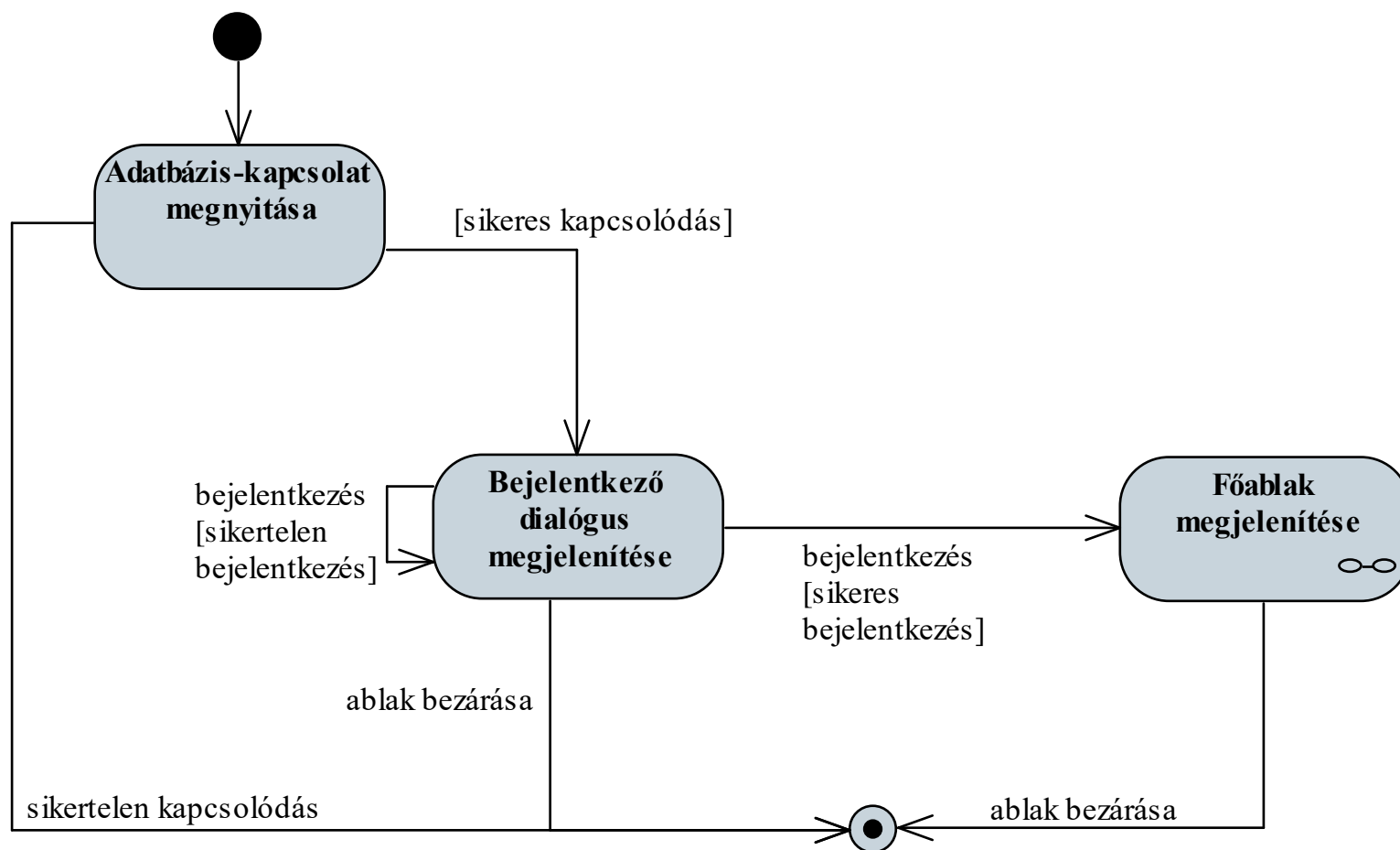
Dinamikus tervezés (adminisztrátor):

- Elsőként be kell jelentkezünk az alkalmazásba a helyes felhasználónév és jelszó megadásával egy dialógusablakban (többször is próbálkozhatunk)
 - ehhez meg kell nyitni az adatbázissal a kapcsolatot
- A főablakban betöltődnek az adatok, majd lehetőségünk van az adatok megtekintésére, szerkesztésére, és mentésére, illetve bezáráskor is mentjük a módosításokat
 - sikertelen mentés esetén hibajelzést kapunk
- Amennyiben nem sikerül megnyitni a kapcsolatot, vagy bezárjuk valamelyik ablakot, az alkalmazás kilép

Esettanulmányok

Utazási ügynökség

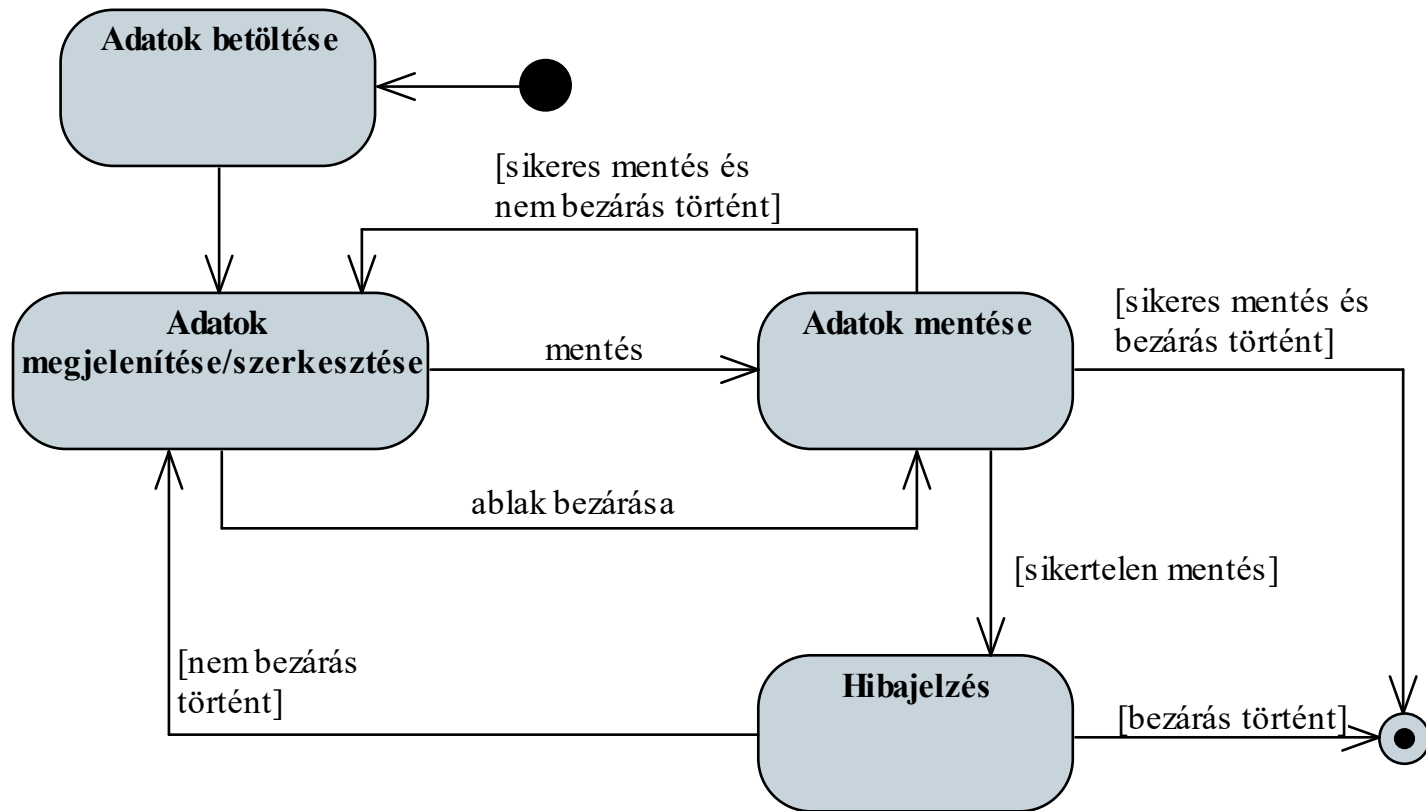
Dinamikus tervezés (adminisztrátor állapotai):



Esettanulmányok

Utazási ügynökség

Dinamikus tervezés (főablak állapotai):



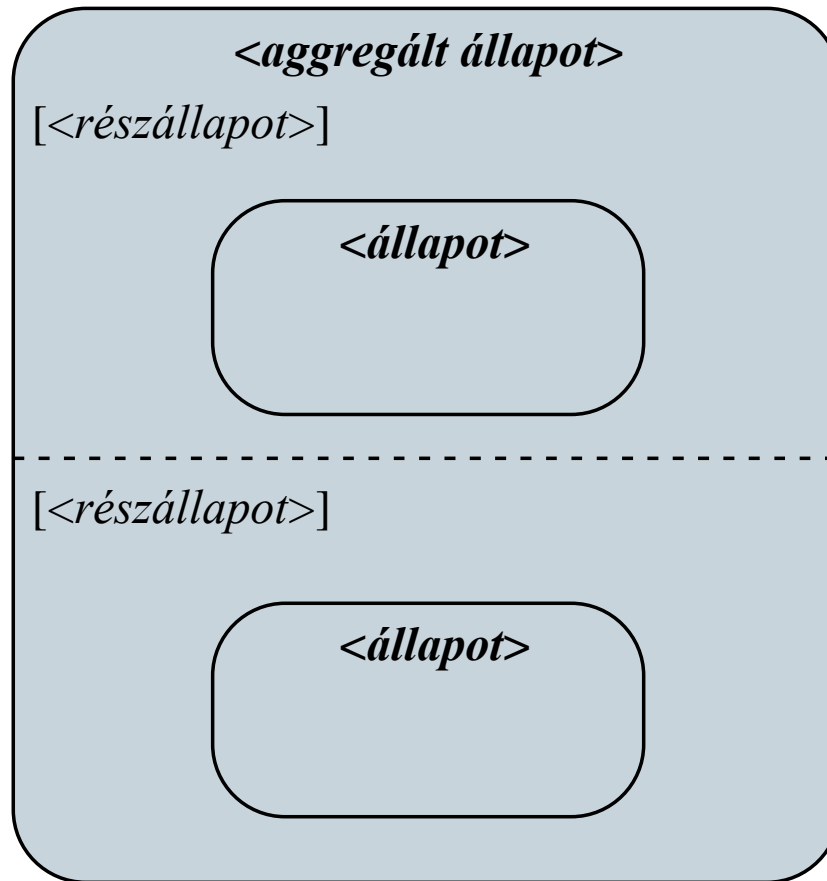
Objektumorientált tervezés: állapotkezelés

Aggregált állapotok

- Egymástól független, párhuzamosan fennálló állapotokat is kezelhetünk a programban, ekkor állapotok *aggregációjáról* (*aggregation*) beszélünk
 - egy osztály állapotait kettő, vagy több szempont szerint állapítjuk meg, és eszerint különböző állapotokat definiálunk
 - egy rendszer állapotát modellezzük, amely a benne szereplő osztályok állapotainak összessége
- Ugyan maguk az állapotok egymástól függetlenül léteznek, mégis hatással lehetnek egymásra tevékenységek, feltételek segítségével
 - pl. ugyanazon tevékenység okozhat párhuzamos állapotváltásokat

Objektumorientált tervezés: állapotkezelés

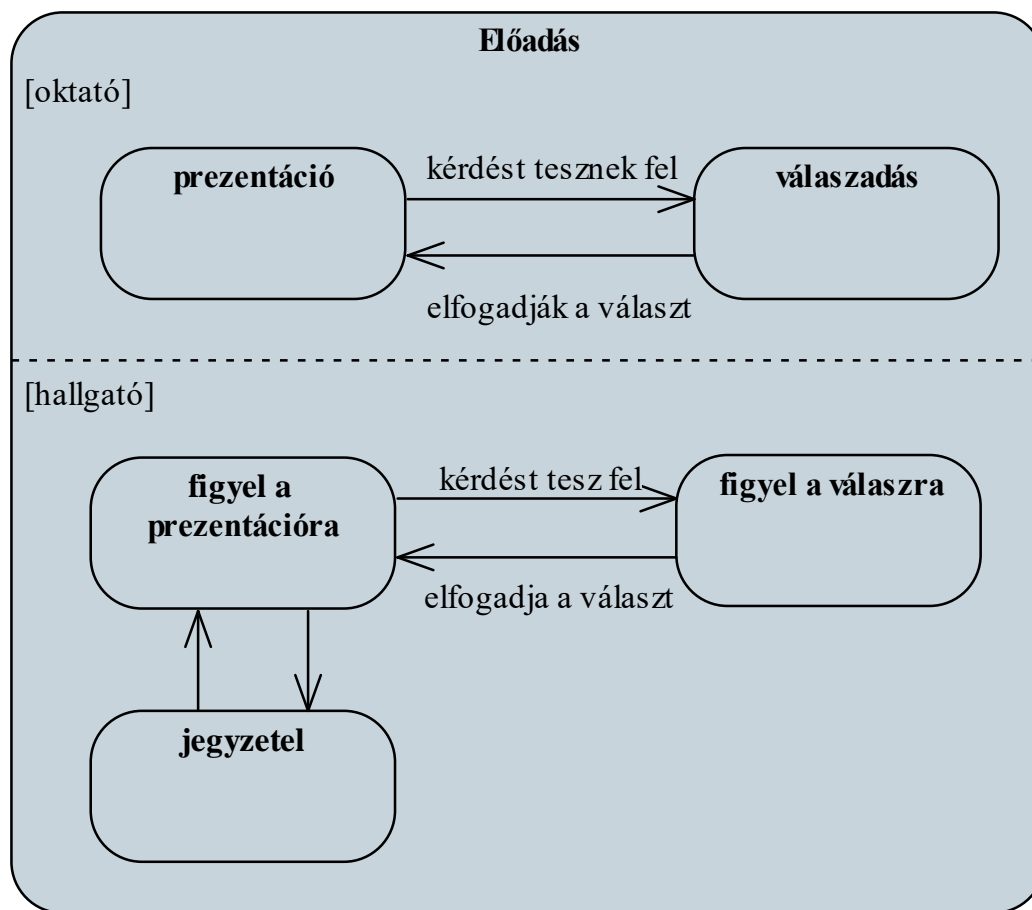
Aggregált állapotok



Objektumorientált tervezés: állapotkezelés

Aggregált állapotok

- Pl. (előadás):



Objektumorientált tervezés: állapotkezelés

Aggregált állapotok

- Állapot aggregáció esetén az állapotnak *párhuzamos részállapotai* (*concurrent substate*) keletkeznek
 - a részállapotok lehetnek állapotátmenetek, általánosított állapotok, rendelkezhetnek kezdeti és végállapottal
 - amennyiben az aggregált állapot rendelkezik kezdeti, vagy végállapottal, akkor minden részállapotnak is rendelkeznie kell vele
 - az aggregált állapot belépési és kilépési eseményét, paramétereit, feltételét és hatását minden részállapot örökli

Esettanulmányok

Memory játék

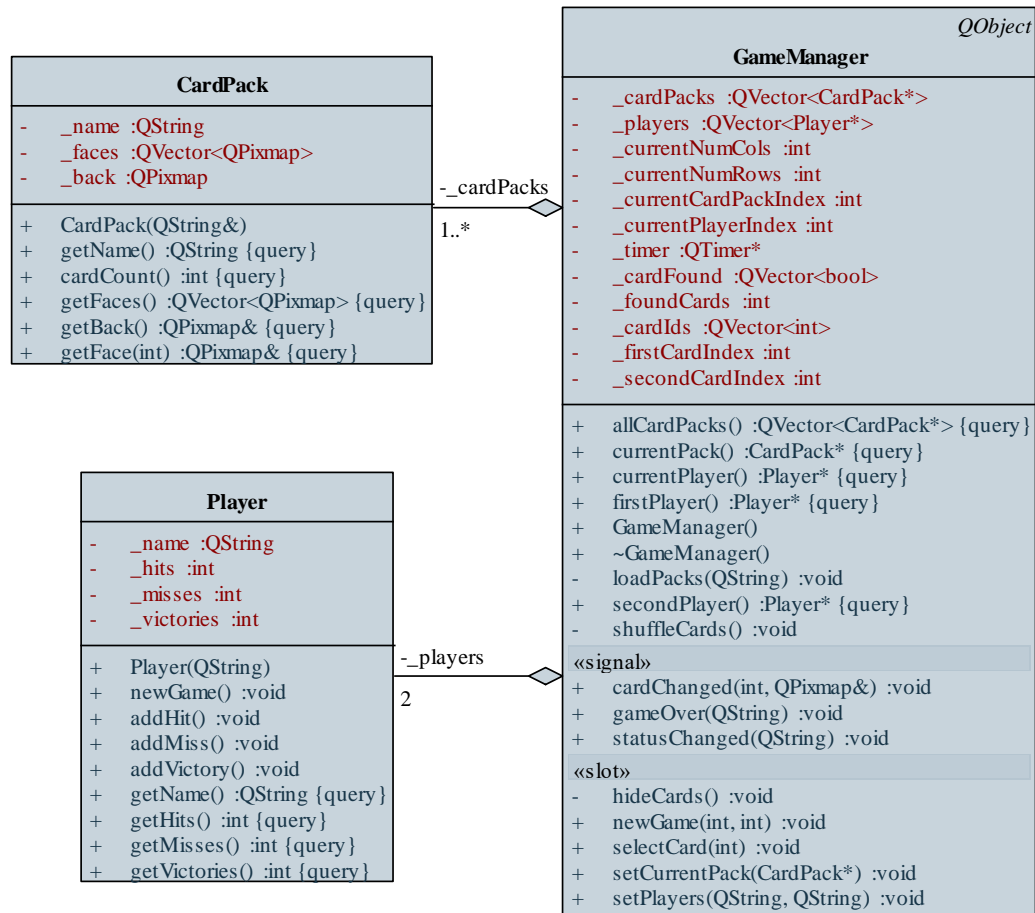
Feladat: Készítsünk egy *Memory* kártyajátékot, amelyben két játékos küzd egymás ellen, és a cél kártyapárok megtalálása a játéktáblán.

- a játékosok felváltva lépnek, minden lépésben felfordíthatnak két kártyát
- amennyiben a kártyák egyeznek, úgy felfordítva maradnak és a játékos ismét léphet, különben visszafordulnak, és a másik játékos következik
- a játékot az nyeri, aki több kártyapárt talált meg
- lehessen a játékosok neveit megadni, kártyacsomagot választani, valamint a kártyák számát (a játéktábla méretét) szabályozni

Esettanulmányok

Memory játék

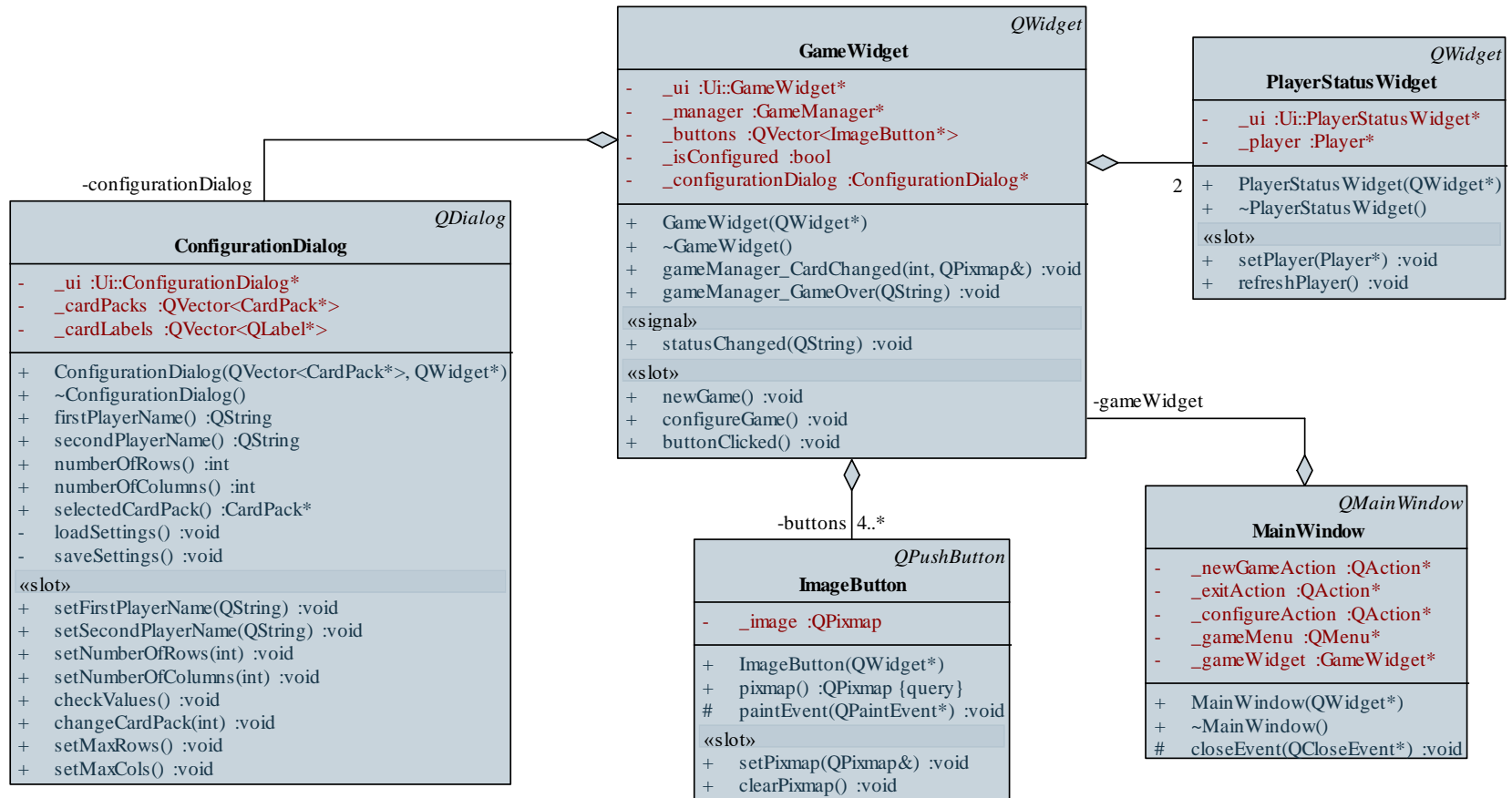
Szerkezeti tervezés (modell):



Esettanulmányok

Memory játék

Szerkezeti tervezés (nézet):



Esettanulmányok

Memory játék

Dinamikus tervezés:

- A főablakot létrehozzuk, majd megjelenítjük (**show**)
 - bezárásához (**closeEvent**) megerősítést kérünk a felhasználótól, amelyben lehetősége van azt megszakítani (**ignore**), vagy elfogadni (**accept**)
- A megjelenítés alatt egymástól függetlenül kezeljük a játék modelljét (**GameManager**), valamint megjelenítését (**GameWidget**)
 - a megjelenítésben fontos különbség, hogy a felület aktív, vagy sem, mivel csak előbbi esetben lehet egérrel kattintani (**mouseClick**)

Esettanulmányok

Memory játék

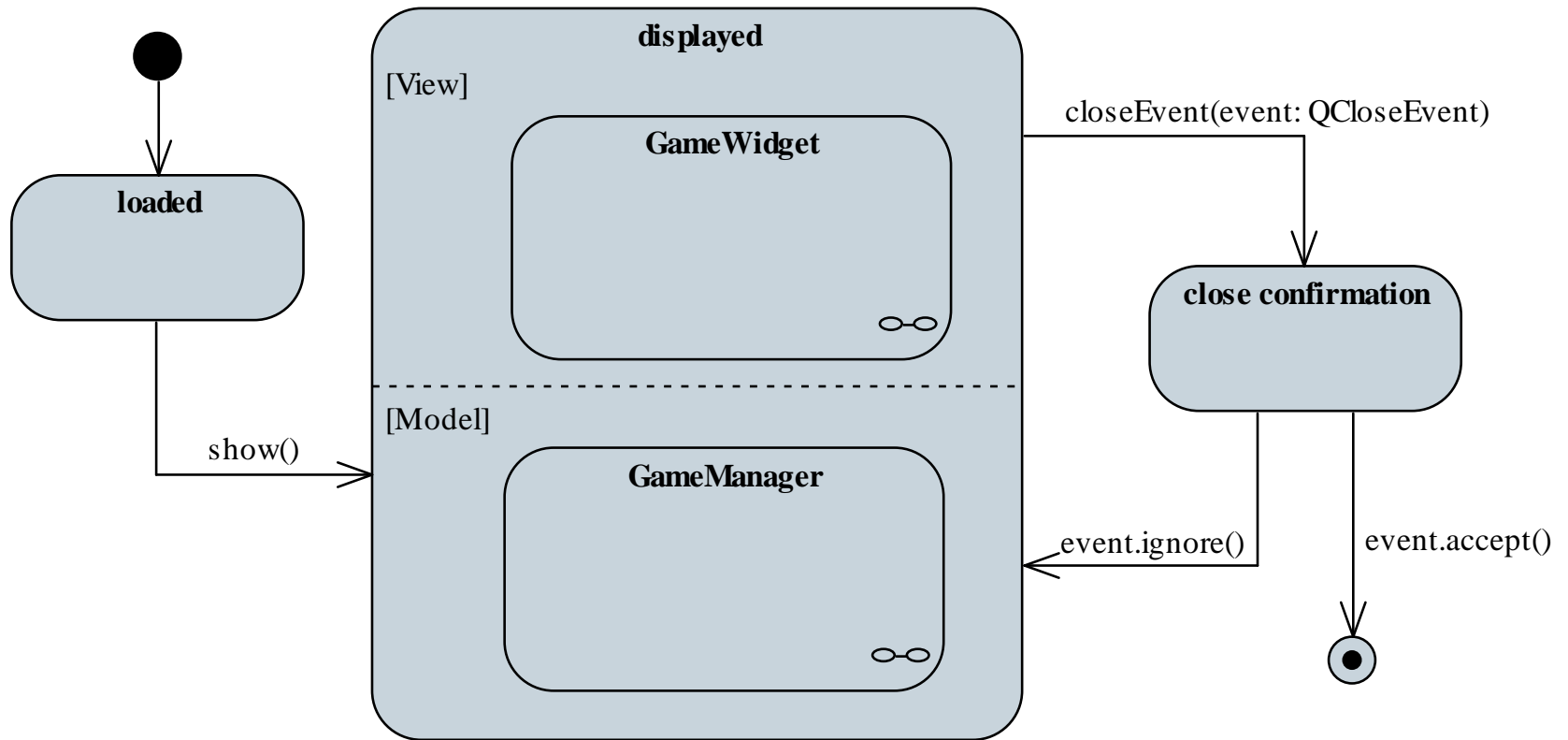
Dinamikus tervezés:

- amennyiben új játékot kezdünk (**newGame**), a felület aktív lesz, játék végén (**gameOver**) pedig inaktívvá válik
- a játék modellje kezdetben egy kártyát sem mutat, de új játék kezdésekor (**newGame**) az összes kártyát megmutatja, majd automatikusan elrejtí őket (**hideCards**)
- kiválasztás (**selectCard**) hatására előbb egyet, majd kettőt megmutathat (**cardChanged**)
- amennyiben a két kártya egyezik, és minden kártyát felfedtünk, vége a játéknak (**gameOver**)

Esettanulmányok

Memory játék

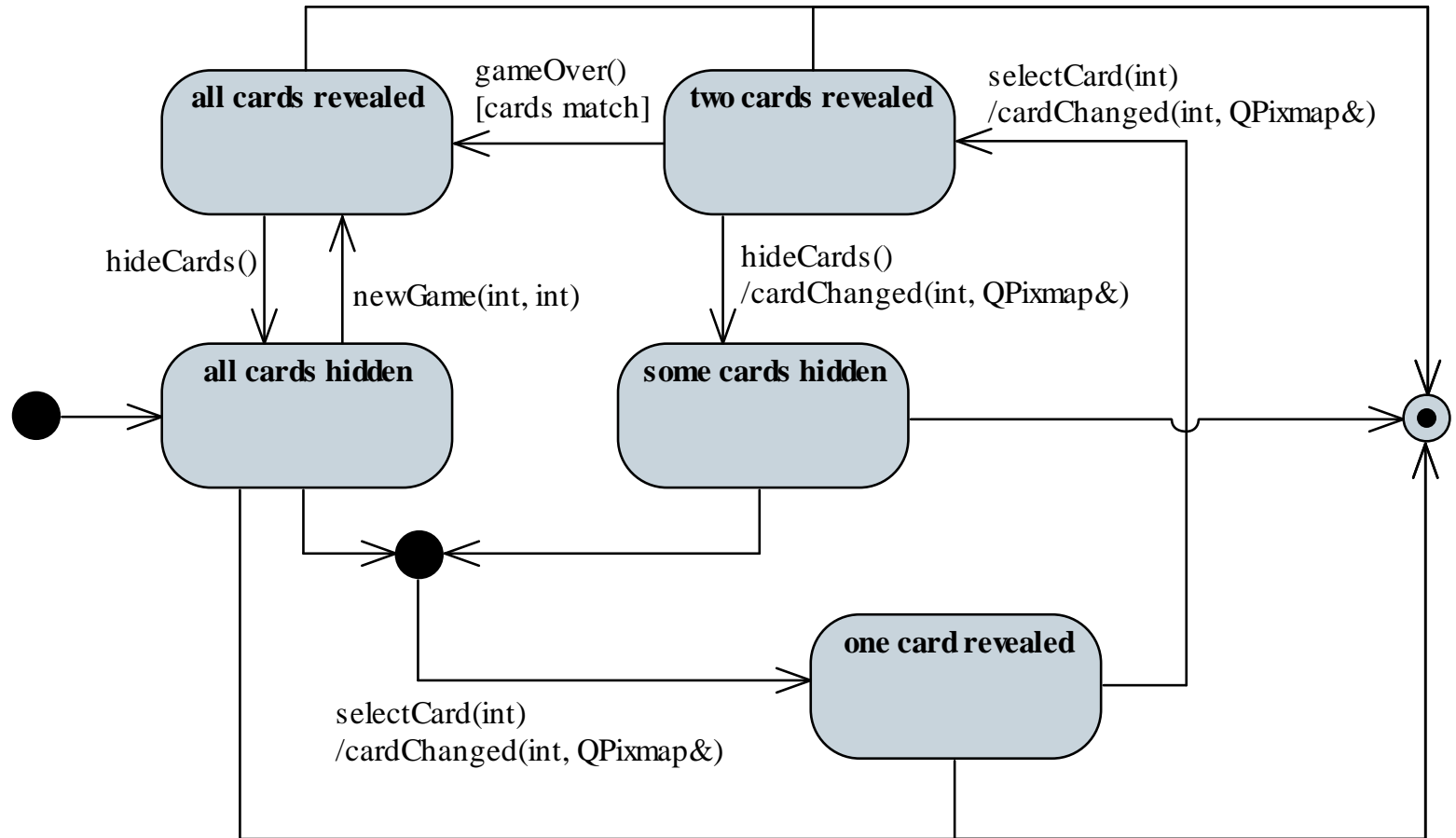
Dinamikus tervezés (alkalmazás állapotai):



Esettanulmányok

Memory játék

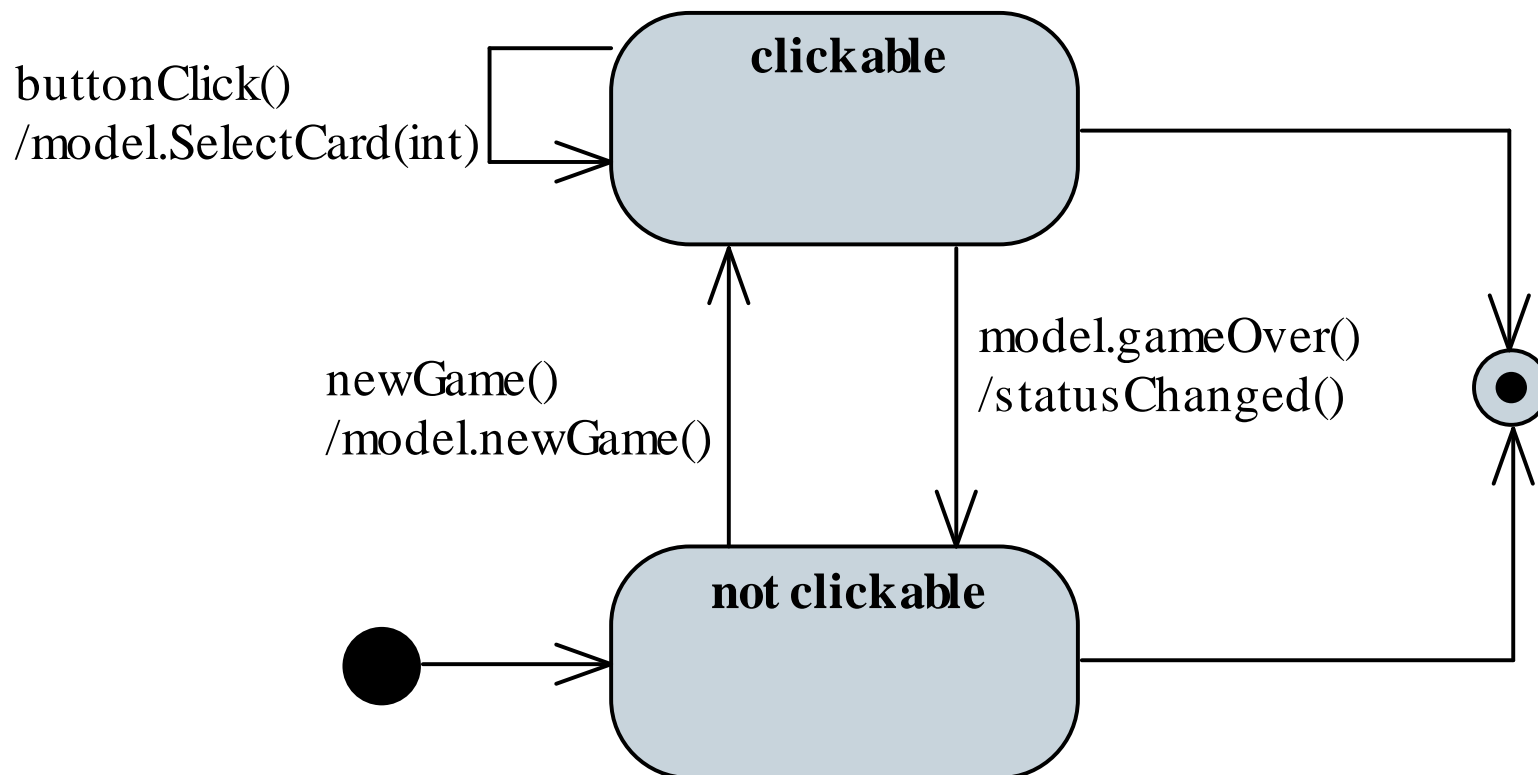
Dinamikus tervezés (GameManager állapotai):



Esettanulmányok

Memory játék

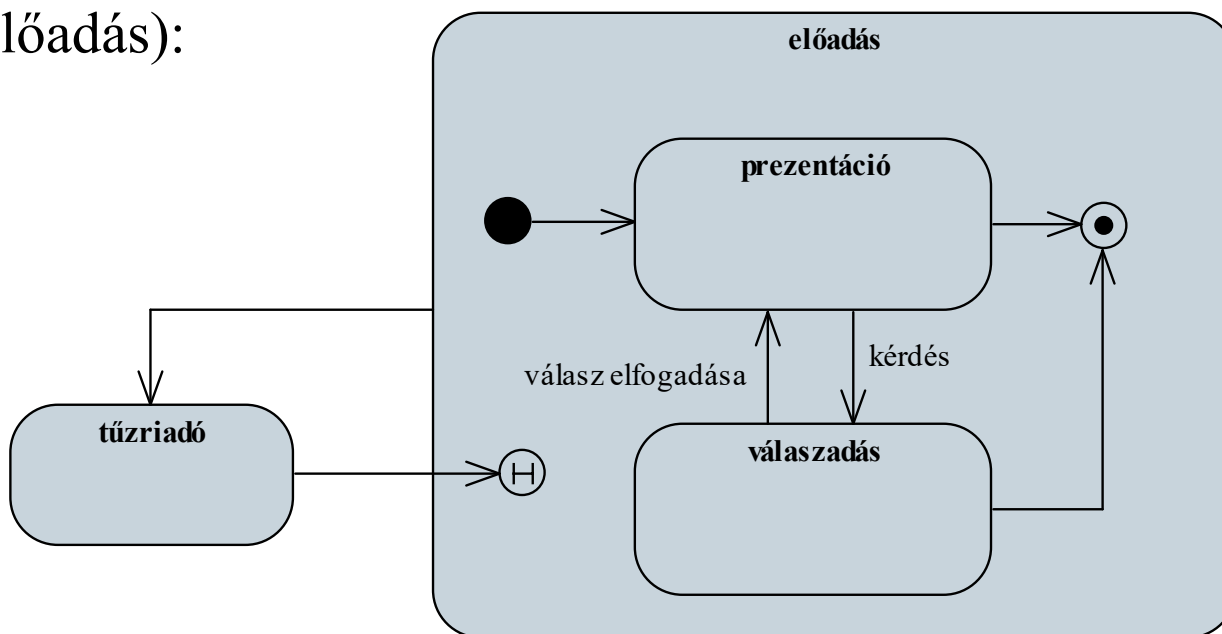
Dinamikus tervezés (GameWidget állapotai):



Objektumorientált tervezés: állapotkezelés

Hisztorizációs állapotok

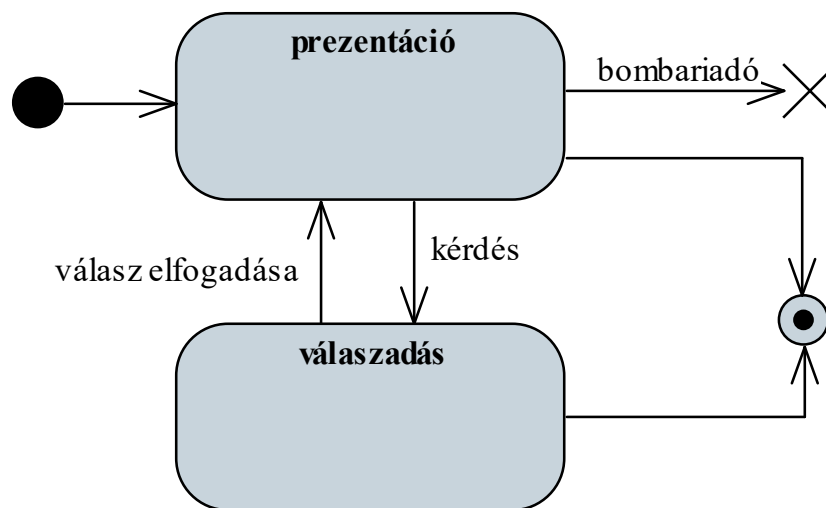
- Általánosított állapotok esetén előfordulhat, hogy a végrehajtás megszakad, és a végrehajtást a megszakítás pontjában szeretnénk végezni, ebben az esetben *hisztorizációs állapotot* (*history state*) használatunk
- Pl. (előadás):



Objektumorientált tervezés: állapotkezelés

Terminálás

- Amennyiben az állapotátmenetet nem a végállapotban szeretnénk bejezni, lehetőségünk van *terminálni* (*terminate*) az állapotautomatát
 - egy szabályos megszakítási pont, amely nem hagyja abnormális állapotban a programot (pl. kivétel kiváltás, párhuzamos futtatás megszakítása)
- Pl. (előadás):



Objektumorientált tervezés: állapotkezelés

Párhuzamos programok állapotai

- Amennyiben az alkalmazás párhuzamosan is tud tevékenységeket végezni, *szétválaszthatjuk (fork)* a végrehajtást, majd később *összefuttathatjuk* a párhuzamos működést (*join*)
- Pl. (előadás):

