Eötvös Loránd Tudományegyetem
Informatikai Kar

# Szoftvertechnológia

## 7. előadás

## Objektumorientált tervezés: végrehajtás

**Giachetta Roberto**

groberto@inf.elte.hu
http://people.inf.elte.hu/groberto

„The good thing about bubbles and arrows, as opposed to programs, is that they never crash.”

(Bertrand Meyer)

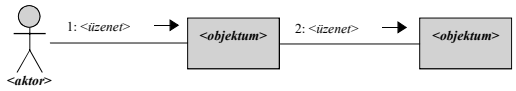


- Az objektumorientált programokat egymással kapcsolatban álló objektumok alkotják
  - a lehetséges kommunikációs pontokat az osztálydiagramban feltérképeztük
  - azonban a végrehajtás sorrendjére, időbeli lefolyására az osztálydiagram nem ad támpontot
  - az állapotdiagram csak egy osztály szemszögéből jellemzi a működést, és elsősorban nem a végrehajtást modellezi
- A program működése során történő, objektumok és osztályok közötti interakciós folyamatokat *kommunikációs*, *szekvencia*, illetve *tevékenység diagram*mal modellezhetjük





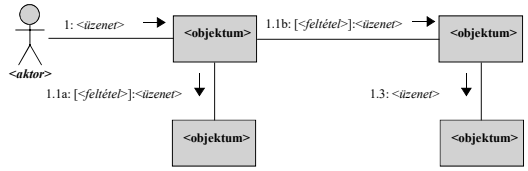
- Az *UML kommunikáció diagram* (*communications diagram*) célja az objektumok közötti kommunikáció sorrendjének megállapítása
  - ábrázolja az objektumokat és a köztük lévő kommunikációt, *üzenetátadás*t (metódushívás, eseménykiváltás)
  - az objektumok mellett szerepeltetheti a rendszer aktorait is, amelyek kezdeményezhetik az üzenetátadást
  - az üzenetekhez rendel irányt és sorrendiséget








- A kommunikációban ábrázolhatjuk
  - a csoportokat, amelyek az egy híváslánchoz tartozó üzenetek (**`<csoport>.<sorszám>`** formátumban)
  - az elágazásokat (**`<sorszám><ág>`** formátumban)
  - a feltételeket (**`[<feltétel>]`** formátumban)



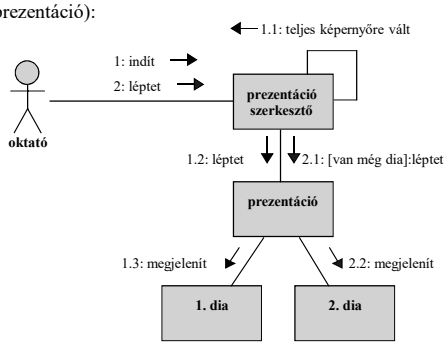





- A kommunikációt az objektumok szemszögéből ábrázoljuk
  - általában nem a teljes rendszer kommunikációját, csak egy leszűkített részét ábrázoljuk, amelyben egy megadott forgatókönyvet követünk
    - pl. egy adott használati eset (funkció) teljesítésének megvalósítását adott feltételek mellett
    - nem tartalmaz feltételt, ciklust, és nem látható az objektumok élettartama
  - segíthet az objektumok viselkedési mintájának meghatározásában (ugyanakkor a pontos ábrázoláshoz szükséges a statikus szerkezet)

## Objektumorientált tervezés: végrehajtás
**Kommunikációs diagram**

- Pl. (prezentáció):






## Objektumorientált tervezés: végrehajtás
**Kommunikációs diagram**

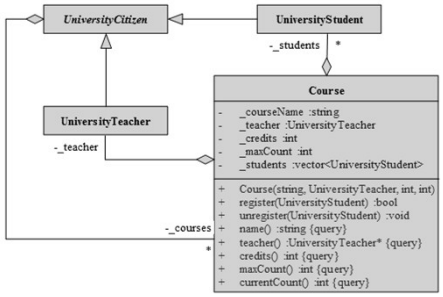
*Feladat*: Készítsünk egy programot, amelyben egyetemi oktatók, hallgatók és kurzusok adatait tudjuk tárolni.

- 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 a **`newCourse`** művelettel,
- a hallgató lejelentkezhet a kurzusról (**`unregister`**), amennyiben jelentkezett már rá (ekkor a kurzust a hallgatótól is elvesszük a **`removeCourse`** művelettel



## Objektumorientált tervezés: végrehajtás
**Kommunikációs diagram**

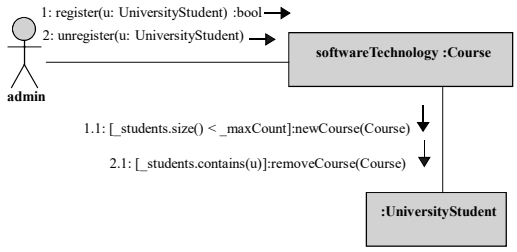
*Szerkezeti tervezés*:






## Objektumorientált tervezés: végrehajtás
**Kommunikációs diagram**

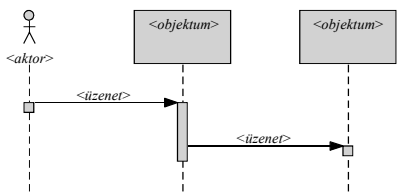
*Dinamikus tervezés* (kommunikáció):






## Objektumorientált tervezés: végrehajtás
**Szekvencia diagram**

- Az *UML szekvencia diagram* (*sequence diagram*) célja az objektumok közötti interakció időrendi ábrázolása
  - tartalmazza a kommunikációs diagram elemeit, ugyanakkor nem sorrendiséget ad a kommunikációra, hanem időbeli lefolyást ábrázol

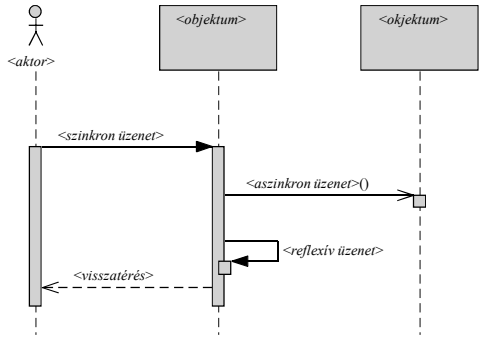





## Objektumorientált tervezés: végrehajtás
**Szekvencia diagram**

- A szekvenciában az objektumok (és az aktorok)
  - *életvonal*lal (*lifeline*) rendelkeznek, amely meghatározza létezésük időtartamát
  - lehetnek *aktív*ak, ekkor képesek kommunikáció kezdeményére
- A szekvenciában az üzeneteknek különböző típusait tudjuk ábrázolni
  - *szinkron üzenet*: feldolgozását (végrehajtása) a hívó megvárja, addig nem végez további műveleteket
  - *aszinkron üzenet*: feldolgozását a hívó nem várja meg, hanem tovább tevékenykedik
  - *visszatérési üzenet*: egy korábbi üzenet feldolgozásának eredménye

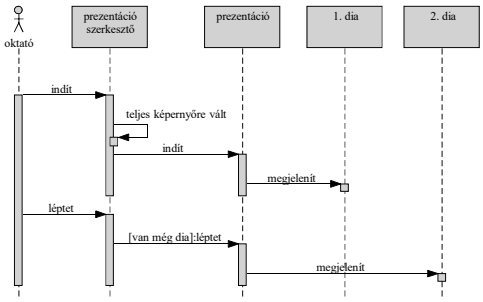
## Objektumorientált tervezés: végrehajtás
### Szekvencia diagram






## Objektumorientált tervezés: végrehajtás
### Szekvencia diagram

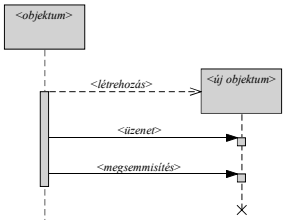
- Pl. (prezentáció):






## Objektumorientált tervezés: végrehajtás
### Objektumok élettartama

- A szekvencia során üzenet segítségével
  - *létrehozhatunk* új objektumokat (a konstruktorral), ekkor elindul az életvonaluk
  - *megsemmisíthetünk* objektumokat (a destruktorral), ekkor vége az életvonaluknak
  - kommunikálhatunk az objektumokkal a két üzenet között



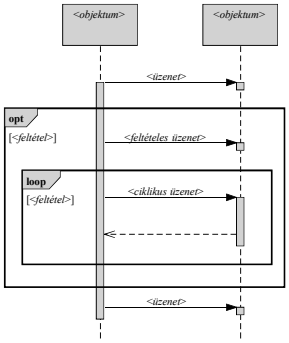



## Objektumorientált tervezés: végrehajtás
### Összetett végrehajtás ábrázolása

- A szekvencia során ábrázolhatunk
  - *feltételes szakaszt* (`opt`), amely csak a feltétel teljesülésekor hajtódik végre
  - *elágazást* (`alt`), ahol a feltétel függvényében különböző ágakat hajthatunk végre
  - *ciklust* (`loop`), ahol a tevékenységet a feltétel függvényében többször is végrehajtjuk
  - *párhuzamos szakaszt* (`par`), ahol egyszerre párhuzamosan végezzük a tevékenységeket
  - *kritikus szakaszt* (`critical`), amely nem végezhető párhuzamosan



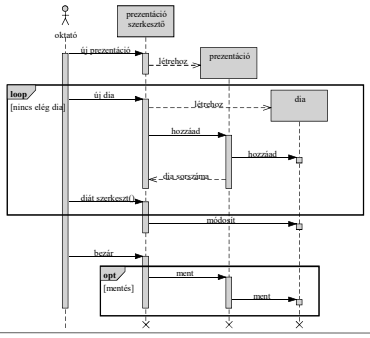
## Objektumorientált tervezés: végrehajtás
### Összetett végrehajtás ábrázolása






## Objektumorientált tervezés: végrehajtás
### Összetett végrehajtás ábrázolása

- Pl. (prezentáció szerkesztés):



## Esettanulmányok
**Tic-Tac-Toe játék**

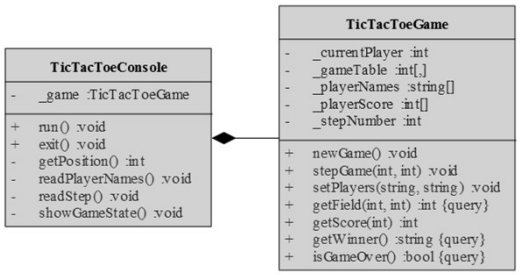
*Feladat*: Készítsünk egy Tic-Tac-Toe programot, amelyben két játékos küzdhet egymás ellen.

- 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
**Tic-Tac-Toe játék**

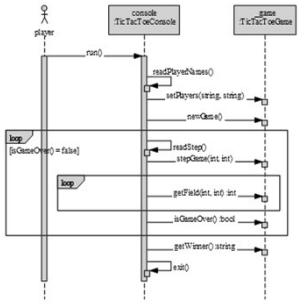
*Dinamikus tervezés* (kommunikáció):

- A játék általános szekvenciája:
  - a játékot futtatjuk (**`run`**)
  - a játékosok megadják neveiket (**`readPlayerNames`**, **`setPlayers`**)
  - elindul a játék (**`newGame`**)
  - a játékosok felváltva lépnek (**`readStep`**, **`stepGame`**)
  - minden lépés közben megjelenítjük az állást (**`showGameState`**, **`getField`**), és ellenőrizzük az állapotot (**`isGameOver`**)
  - amennyiben vége van a játéknak, lekérdezzük a győztes nevét (**`getWinner`**)



## Esettanulmányok
**Tic-Tac-Toe játék**

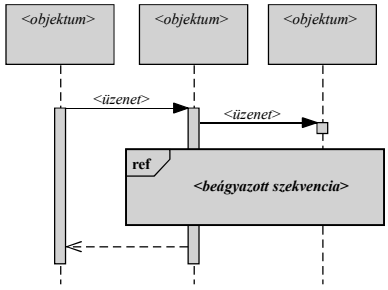
*Dinamikus tervezés* (kommunikáció):






## Objektumorientált tervezés: végrehajtás
**Beágyazott diagramok**

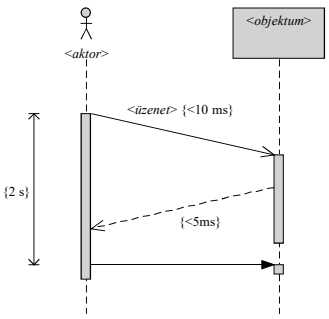
- Egy szekvenciába beágyazhatunk másik szekvenciát is (**`ref`**), ezzel csökkentve a diagram bonyolultságát






## Objektumorientált tervezés: végrehajtás
**Időtartam**

- A szekvenciában az idő relatív lefolyását tényleges időbeli lefolyássá is alakíthatjuk időtartam megadásával
  - üzenetek és üzenetek közötti szakaszok rendelkezhetnek időtartammal
  - az időtartam lehet pontos, intervallum, vagy korlát (alsó/felső)



## Esettanulmányok
**Marika néni kávézója**

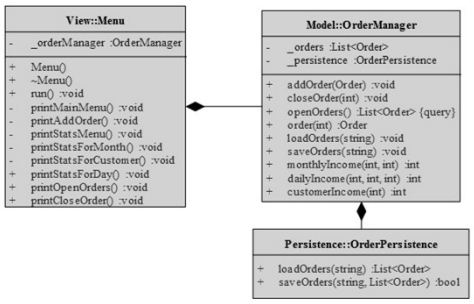
*Feladat*: Készítsük el Marika néni kávézójának eladási nyilvántartását végigkövető programot.

- 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 törzsvásárolói számra összesített nettó/bruttó fogyasztási statisztikák követésére



## Esettanulmányok
**Marika néni kávézója**

*Szerkezeti tervezés:*






## Esettanulmányok
**Marika néni kávézója**

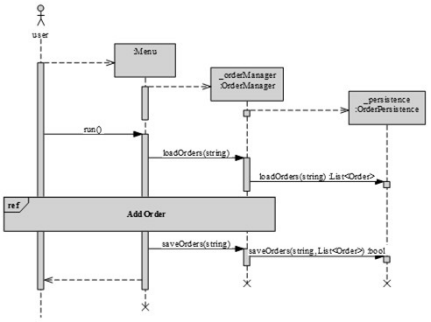
*Dinamikus tervezés*:

- Egy tétel hozzáadásának szekvenciája:
  - futtatjuk a menüt (**`run`**), amely először betölti az adatokat (**`loadOrders`**)
  - rendelés létrehozásakor (**`printAddOrder`**) felvesszük a tételeket (**`addItem`**), majd elmentjük a rendelést (**`addOrder`**)
  - listázva a nyitott rendeléseket (**`printOpenOrders`**) van lehetőségünk lezárni egy rendelést (**`printCloseOrder`**, **`closeOrder`**)
  - a futásból történő kilépéskor elmentjük az adatokat (**`saveOrders`**)



## Esettanulmányok
**Marika néni kávézója**

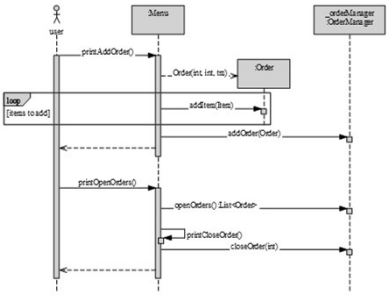
*Dinamikus tervezés* (rendelés felvétele, kommunikáció):






## Esettanulmányok
**Marika néni kávézója**

*Dinamikus tervezés* (**`Add Order`**):






## 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 egyeznek, úgy felfordítva maradnak és a játékos ismét léphet, különben 1 másodperc múlva visszafordulnak
- a játékot az nyeri, aki több kártyapárt talált meg
- lehessen a játékosok neveit megadni, kártyacsomagot választani, valamint a kártyák számát (a játéktábla méretét) szabályozni

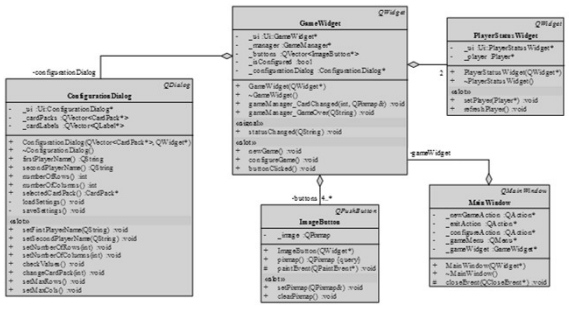
## Esettanulmányok
**Memory játék**

*Szerkezeti tervezés (modell):*




## Esettanulmányok
**Memory játék**

*Szerkezeti tervezés (nézet):*






## Esettanulmányok
**Memory játék**

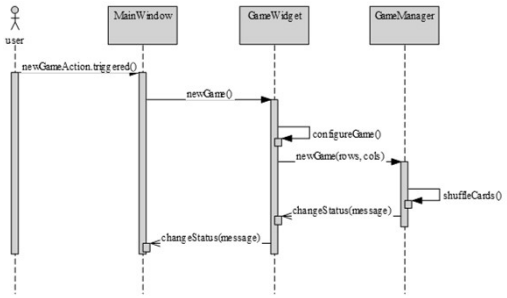
*Dinamikus tervezés*:

- Új játék indításának szekvenciája:
  - új játék indításához először a főablakban (**`MainWindow`**) kell kiváltanunk (**`triggered`**) a megfelelő akciót (**`newGameAction`**)
  - ennek hatására a főablak új játékot indít (**`newGame`**) a játék nézetében (**`GameWidget`**)
  - a nézet beállítja a játék paramétereit (**`configureGame`**)
  - a nézet létrehozza az új játékot (**`newGame`**) a modellben (**`GameManager`**)
  - a modell megkeveri a kártyákat (**`shuffleCards`**), majd eseménnyel jelzi az állapot változását (**`changeStatus`**)



## Esettanulmányok
**Memory játék**

- *Dinamikus tervezés* (új játék indítása, kommunikáció):





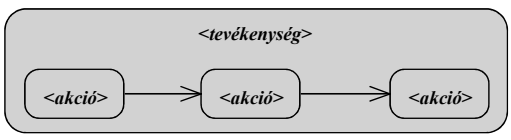

## Objektumorientált tervezés: végrehajtás
**Tevékenység diagram**

- A kommunikációs és szekvencia diagramok az interakciót elsősorban az objektumok szempontjából közelítik meg, nem a végrehajtott tevékenységsorozat szemszögéből
- Az *UML tevékenység diagram* (*activity diagram*) célja, hogy a végrehajtás lefolyását a tevékenységek és a tevékenységekben felhasznált adatok szempontjából közelítse meg
  - egy időbeli lefolyását látjuk a kommunikációnak, de a kommunikáció végrehajtója rejtett marad
  - jelölésrendszere hasonlít az állapotdiagramra, ugyanakkor jelentésében közelebb áll a szekvenciadiagramhoz



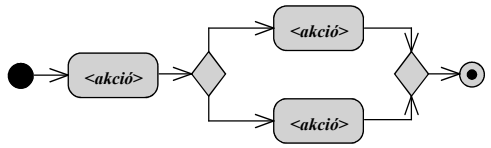
## Objektumorientált tervezés: végrehajtás
**Tevékenység diagram elemei**

- A tevékenység diagram egy *tevékenység*et (*activity*) ábrázol, amely egy munkafolyamat megvalósulása
  - a munkafolyamat egyes lépései az *akció*k (*action*), amelyek adott funkciók végrehajtásai
  - az akciókat a *vezérlési folyam* (*control flow*) köti össze, amely meghatározza sorrendjüket



## Objektumorientált tervezés: végrehajtás
### A tevékenység diagram elemei

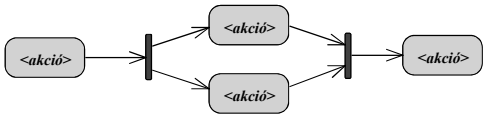
- A tevékenység során
  - egy kezdeti állapotból (*initial*) egy végállapotba (*final*) vezetjük a vezérlési folyamot
  - elágazhatunk adott feltételek mentén (*decision*) a végrehajtásban, illetve különböző ágakat összevonhatunk (*merge*)



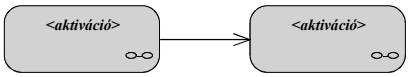



## Objektumorientált tervezés: végrehajtás
### A tevékenység diagram elemei

- párhuzamosíthatunk (*fork*), valamint összefuttathatjuk (*join*) a párhuzamos végrehajtást



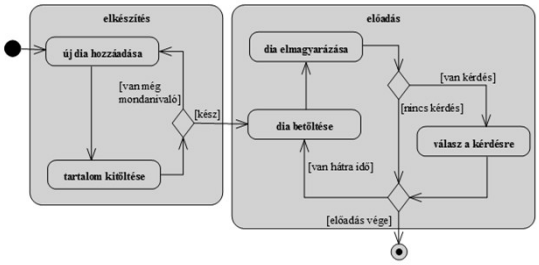

- általánosíthatjuk a tevékenységet (*generalization*, *composite*)






## Objektumorientált tervezés: végrehajtás
### A tevékenység diagram elemei

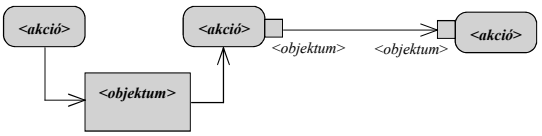
- Pl. (prezentáció elkészítése és előadása):






## Objektumorientált tervezés: végrehajtás
### Adatok a tevékenységben

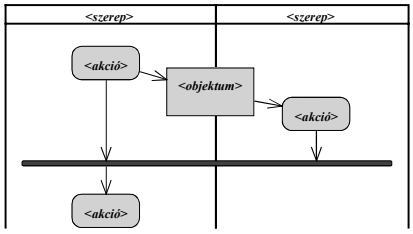
- Ábrázolhatjuk a tevékenység során átadott adatokat (objektumokat), amelynek két lehetősége:
  - az átadott/átvett objektum beiktatása a vezérlési folyamba
  - az átadott, illetve átvett objektum jelölése az akciónál (mint az akció bemenő, illetve kimenő értékei)






## Objektumorientált tervezés: végrehajtás
### Tevékenységek felosztása

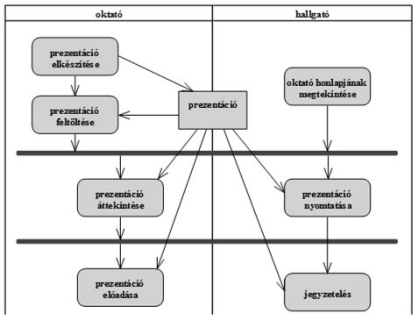
- Amennyiben azonosítani szeretnénk a tevékenységben betöltött szerepeket, feloszthatjuk a tevékenységet párhuzamos folyamokra (*partitions*), amelyek között szinkronizálhatunk (*syncronization*)






## Objektumorientált tervezés: végrehajtás
### Tevékenységek felosztása

- Pl. (prezentáció elkészítése és előadása):



## Esettanulmányok
**Marika néni kávézója**

*Dinamikus tervezés* (tevékenység):

- Az alkalmazás indításakor betöltjük az adatokat, majd használjuk a menüt, végül mentjük az adatokat


- Lehetőségünk van új rendelés feltételére, amelyben létrehozunk egy új rendelést (esetlegesen törzsvásárlói kártya megadásával), tételeket veszünk fel, majd lezárjuk a rendelést (ha a vendég fizetett)



## Esettanulmányok
**Marika néni kávézója**

*Dinamikus tervezés* (tevékenységek):