Eötvös Loránd Tudományegyetem
Informatikai Kar

Szoftvertechnológia

5. előadás

Objektumorientált tervezés: architektúra

Giachetta Roberto

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

„Architecture is about the important stuff. Whatever that is.”

(Martin Fowler)

Objektumorientált tervezés: architektúra
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 pontosítjuk a már létező osztályok felépítését, az implementációs megkötéseket
    - felbonthatunk osztályokat, amennyiben túl bonyolulttá, túl szerteágazóvá válnak
    - összevonhatunk osztályokat, amennyiben túlzottan elaprózódnak

Objektumorientált tervezés: architektúra
A tervezés elősegítése

- Az objektumok és osztályok azonosításában segítenek
  - az objektumorientált tervezés általános elgondolásai (egységbe zárás, öröklődés, …)
  - az objektumorientált tervezés alapelvei (*SOLID* elvek), amelyek általános elvárásokat írnak le
  - a tervezési minták, vagy *tervminták* (*design patterns*), amelyek jól ismert problémakörökre adnak általánosan megfogalmazott megoldást
    - általában egy objektumorientált szerkezetet definiálnak
    - a problémakört visszavezetjük a mintára, és megfeleltetéseket teszünk (mint a programozási tételeknél)

Objektumorientált tervezés: architektúra
SOLID elvek

- *Single responsibility principle* (SRP): egy programegység csak egyvalamiért felelhet
  - minden komponens, osztály, metódus csak egy felelősségi körrel rendelkezzen, ami megváltoztatásának oka lehet
  - így a változtatások csak kis részét érintik a programnak
  - sok jól ismert felelősségi kör adott (megjelenítés, adatkezelés, eseménynaplózás, hálózati kapcsolat, …)
- *Open/closed principle* (OCP): a programegységek nyitottak a kiterjesztésre, de zártak a módosításra
  - új szolgáltatások hozzáadása ne igényelje a jelenlegi programegységek átírását, inkább újak bevezetését

Objektumorientált tervezés: architektúra
SOLID elvek

- *Liskov substitution principle* (LSP): az objektumok felcserélhetőek altípusaik példányára a program viselkedésének befolyása nélkül
  - minden altípusnak biztosítania kell az ős funkcionalitását azok feltételeinek betartása mellett
- *Interface segregation principle* (ISP): nagy, általános interfészek helyett több, kisebb interfészt használjunk
  - így az interfészt megvalósító osztály használója nem függ általa nem igényelt funkcióktól
- *Dependency inversion principle* (DIP): függőségeket csak az absztrakciók között állítunk fel, és nem a konkrét megvalósítások között

Objektumorientált tervezés: architektúra
Függőségek

- Az objektumok között kapcsolatokat definiálhatunk, ez által *függőség*eket hozunk létre
  - egy osztály függ egy másik osztálytól, amennyiben bármilyen módon felhasználja azt
- A függőségeket úgy kell kialakítanunk, hogy
  - az azonos/hasonló feladatot ellátó osztályok között szoros kapcsolat, nagy fokú együttműködés legyen
  - a különböző feladatot ellátó osztályok között laza kapcsolat, kevés együttműködés legyen
- Ez nagyban elősegíti a program modularitását, így a későbbi módosíthatóságot, bővíthetőséget

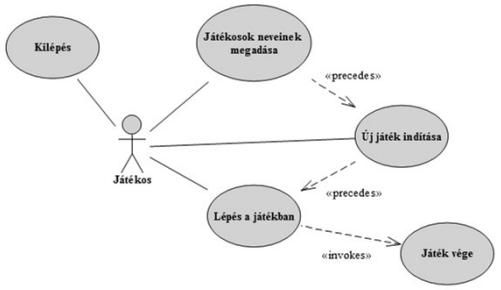
Esettanulmányok
Tic-Tac-Toe játék

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

Használati esetek:



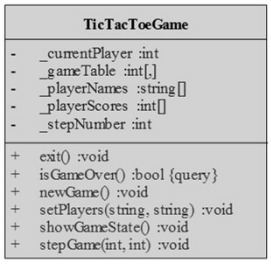
Esettanulmányok
Tic-Tac-Toe játék

Szerkezeti tervezés:

- Egy játékot kell kezelnünk (`TicTacToeGame`), ahol lehet új játékot kezdeni (`newGame`), lépni a játékban (`stepGame`), és vége lehet a játéknak (`isGameOver`)
- Ezen felül a felhasználó láthatja a játékállást (`showGameState`), megadhatja a nevét (`setPlayers`), és kiléphet a játékból (`exit`)
- A játéktábla megfelel egy mátrixnak (`gameTable`), amelyen három különböző állapotot kell megkülönböztetnünk (egész számmal megtehető)
- Nyilvántarthatjuk a játékosok nevét (`playerNames`), az aktuális játékost (`currentPlayer`), illetve a lépésszámot (`stepNumber`), valamint a pontszámokat (`playerScores`)

Esettanulmányok
Tic-Tac-Toe játék

Szerkezeti tervezés:



Esettanulmányok
Tic-Tac-Toe játék

Szerkezeti tervezés:

- A játékot el kell helyeznünk egy konzolos alkalmazásban, ahol futtatjuk a játékot, beolvassuk a felhasználói bevitelt, és megjelenítjük az aktuális állást
  - a felhasználói interakció és a megjelenítés olyan funkciók, amelyek magához a játékhoz kötődnek, egy külön felelősségi kört képeznek, ezért külön osztályban kell megvalósítani őket (SOLID)
    - ha megváltoznak a játékszabályok, akkor a játékot módosítjuk
    - ha megváltozik a megjelenítés módja, az alkalmazást módosítjuk

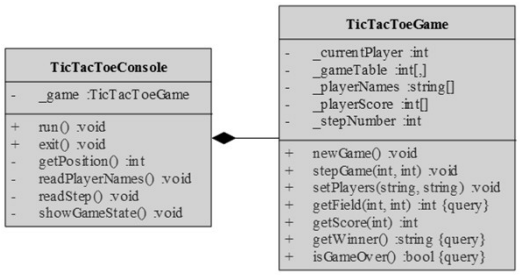
Esettanulmányok
Tic-Tac-Toe játék

Szerkezeti tervezés:

- A bevitellel és megjelenítéssel kapcsolatos funkciókat egy konzolkezelő osztályba helyezzük (`TicTacToeConsole`)
  - futtatja a játékot (`run`), megjeleníti a játékállást (`showGameState`), beolvassa a lépést (`readStep`), a játékosneveket (`readPlayerNames`) és lehetőséget ad a kilépésre (`exit`)
  - így magának a játéknak a bevitellel/megjelenítéssel nem kell foglalkoznia
  - ehhez lehetőséget kell adni a játék elérésére (`game`), valamint a játék állapotának lekérdezésére (`getField`, `getScore`, `winnerName`)

Esettanulmányok
Tic-Tac-Toe játék

Szerkezeti tervezés:



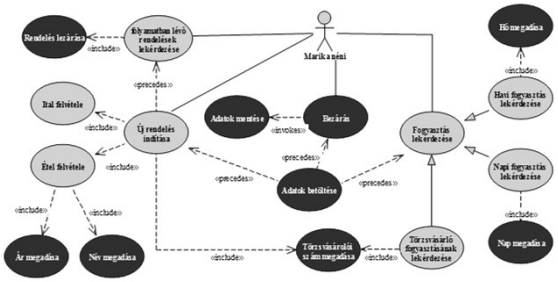
Esettanulmányok
Marika néni kávézója

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

Használati esetek:



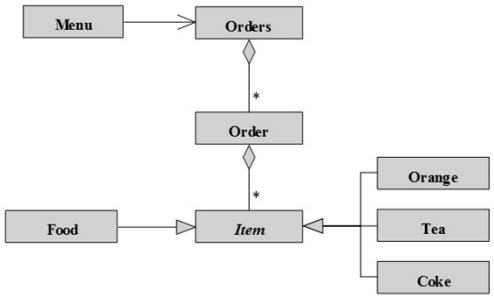
Esettanulmányok
Marika néni kávézója

Szerkezeti tervezés:

- A programban rendeléseket kezelünk, amelyek tetszőleges sok tételből állhat
  - a tételek a hamburger, ufó, palacsinta, kóla, narancs, tea, amelyek mind nagyon hasonlóak, csak néhány részletben térnek el
  - a rendelésekhez tartozható törzsvásárlói szám, illetve lehet lezárt, vagy folyamatban lévő
- Rendelések sorozatával dolgozunk, amelyek száma folyamatosan bővül, a rendeléseket betölthetjük, és menthetjük
- A felhasználói interakciót egy menün keresztül biztosítjuk, amely megjeleníti a tartalmat, és fogadja a felhasználói bevitelt

Esettanulmányok
Marika néni kávézója

Szerkezeti tervezés:



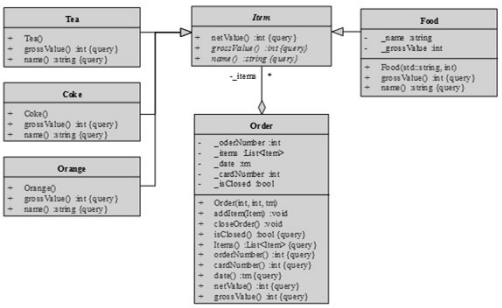
Esettanulmányok
Marika néni kávézója

Szerkezeti tervezés:

- Tétel (`Item`):
  - minden esetben ismert a név, a bruttó és a nettó ár, ami könnyen számolható a bruttóból
  - az italoknál ezek az adatok típustól függenek (így nem kell tárolnunk őket), ételek esetén változhatnak
- Rendelés (`Order`):
  - adatai a sorszám (ez automatikusan generált), a törzsvásárlói szám és a dátum, valamint az állapota (lezárt-e)
  - lehetőséget ad új tétel felvételére, nettó/bruttó érték lekérdezésére
  - a tételeket kezelhetjük egy listában

Esettanulmányok
Marika néni kávézója

Szerkezeti tervezés:



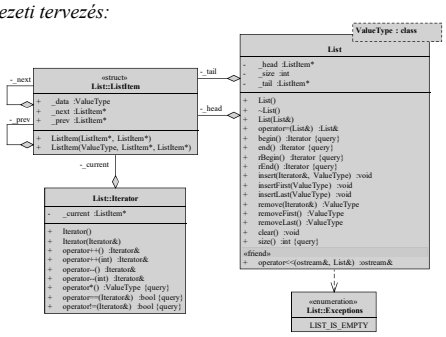
Esettanulmányok
Marika néni kávézója

Szerkezeti tervezés:

- Láncolt lista (`List`):
  - kétszeresen láncolt, fejelemes, aciklikus reprezentáció, sablonos típusként
  - 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
  - a listaelemet és a lista kivételeit beágyazott osztályként hozzuk létre, a listaelem egyszerűsége miatt lehet rekord

Esettanulmányok
Marika néni kávézója

Szerkezeti tervezés:



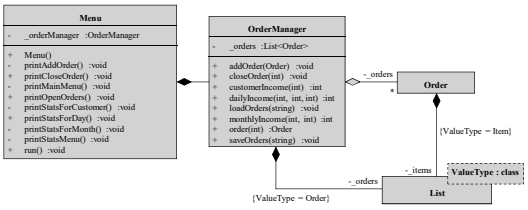
Esettanulmányok
Marika néni kávézója

Szerkezeti tervezés:

- Rendeléskezelő (`Orders` helyett `OrderManager`):
  - kezeli a rendelések listáját, lehet felvenni (`addOrder`), és lekérdezni (`order`) rendeléseket
  - biztosítja a statisztikák lekérdezését (`monthlyIncome`, `dailyIncome`, …)
  - lehetőséget ad adatok betöltésére (`loadOrders`), mentésére (`saveOrders`)
- Menü (`Menu`):
  - biztosítja a futtatás lehetőségét (`run`)
  - több menüpontra tagolódik (`printMainMenu`, …)

Esettanulmányok
Marika néni kávézója

Szerkezeti tervezés:

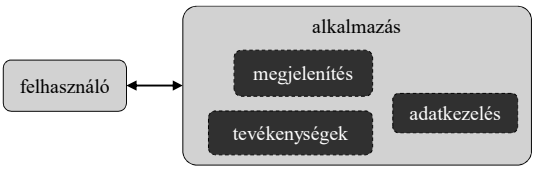


Objektumorientált tervezés: architektúra
Az architektúra

- A szerkezeti (statikus) tervezés leghangsúlyosabb része objektumok, illetve osztályok megtervezése, azonban ez csak egy szempontját jelenti a tervnek
  - eleve az osztályok közvetlen meghatározása egy összetett feladat esetén nehézkes lehet
- *Szoftver architektúrá*nak nevezzük a szoftver fejlesztése során meghozott *elsődleges tervezési döntések* halmazát
  - meghatározzák a rendszer magas szintű felépítését és működését, az egyes alkotóelemek csatlakozási pontjait
  - megváltoztatásuk később jelentős újratervezését igényelné a szoftvernek

Objektumorientált tervezés: architektúra
A monolitikus architektúra

- A legegyszerűbb felépítést a monolitikus *architektúra* (*monolithic architecture*) adja
  - nincsenek programegységekbe szétválasztva a funkciók
  - a felületet megjelenítő kód vegyül az adatkezeléssel, a tevékenységek végrehajtásával, stb.



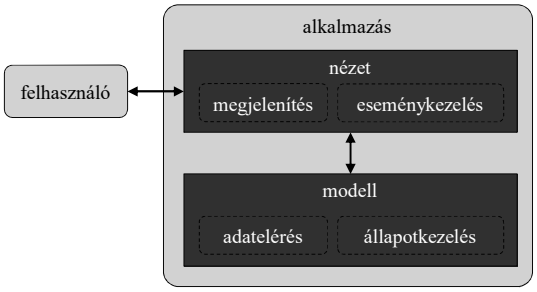
Objektumorientált tervezés: architektúra
A kétrétegű architektúra

- Összetettebb alkalmazásoknál az egyrétegű felépítés korlátozza a program
  - áttekinthetőségét, tesztelhetőségét (pl. nehezen látható át, hol tároljuk a számításokhoz szükséges adatokat)
  - módosíthatóságát, bővíthetőségét (pl. nehezen lehet a felület kinézetét módosítani)
  - újrafelhasználhatóságát (pl. komponens kiemelése és áthelyezése másik alkalmazásba)
- A legegyszerűbb felbontás a felhasználói kommunikáció (megjelenítés, bemenet) leválasztása a tényleges funkcionalitástól, ezt nevezzük *kétrétegű*, *modell/nézet* (*MV*, *model/view*) architektúrának

Objektumorientált tervezés: architektúra
A kétrétegű architektúra

- A modell/nézet architektúrában
  - a *modell* tartalmazza a háttérben futó logikát, azaz a tevékenységek végrehajtását, az állapotkezelést, valamint az adatkezelést, ezt nevezzük *alkalmazáslogiká*nak, vagy *üzleti logiká*nak
  - a *nézet* tartalmazza a grafikus felhasználói felület megvalósítását, beleértve a vezérlőket és eseménykezelőket
  - a felhasználó a nézettel kommunikál, a modell és a nézet egymással
  - a modell nem függ a nézettől, függetlenül, önmagában is felhasználható, ezért könnyen átvihető másik alkalmazásba, és más felülettel is üzemképes

Objektumorientált tervezés: architektúra
A kétrétegű architektúra

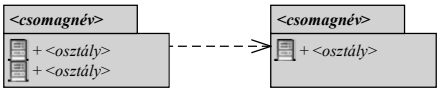


Objektumorientált tervezés: architektúra
Csomagok

- A program szerkezetét *csomagok*ba (*package*) szervezhetjük, ahol a csomag olyan része az alkalmazásnak, amely
  - egy adott feladatcsoporthoz tartozó funkciókat biztosítja, de függhet más csomagoktól
  - a csomagokat és függőségeket irányított gráfban ábrázolhatjuk, amelynek körmentesnek kell lennie (DAG) annak érdekében, hogy megfelelő modularitással rendelkezzen a szoftver

Objektumorientált tervezés: architektúra
A csomagdiagram

- A *csomagdiagram* (*package diagram*) célja a rendszer felépítése a logikai szerkezet mentén, azaz az egyes 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: architektúra
A csomagdiagram

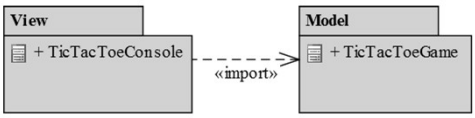
- *használat* (`use`): a csomag felhasznál egy másikat
- *beágyazás* (`nesting`): a csomag egy másiknak a része
- *importálás* (`import`): a csomag betölti a másikat
- *összeillesztés* (`merge`): a csomag tartalmazza, és kibővíti a másik teljes funkcionalitását

- a csomagok az osztálydiagramban is feltüntethetőek

Esettanulmányok
Tic-Tac-Toe játék

Szerkezeti tervezés:

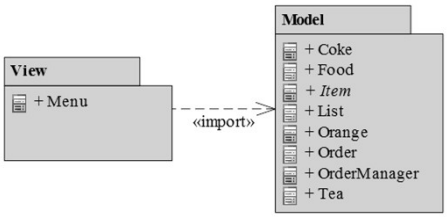
- Az alkalmazásban az `TicTacToeConsole` osztály biztosítja a nézetet, míg a `TicTacToeGame` osztály pedig az üzleti logikát



Esettanulmányok
Marika néni kávézója

Szerkezeti tervezés:

- Az alkalmazásban a `Menu` osztály biztosítja a nézetet, míg a további osztályok a funkcionalitását látják el, azért azok a modellt alkotják



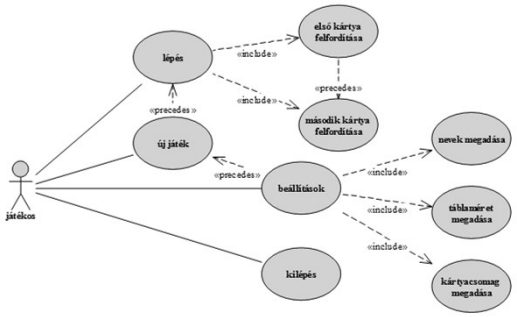
Esettanulmányok
Memory játék

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

Használati esetek:



Esettanulmányok
Memory játék

Szerkezeti tervezés:

- Az alkalmazást modell/nézet architektúrában valósítjuk meg
- A modell tartalmazza:
  - magát a játékot, amit egy kezelőosztály felügyel (`GameManager`), valamint hozzá segédosztályként a játékos adatait (`Player`)
  - a kártyacsomagokat (`CardPack`)
- A nézet tartalmazza:
  - a játék főablakát (`MainWindow`), amely tartalmaz egy menüt és egy státuszsort
  - a beállítások segédablakát (`ConfigurationDialog`)

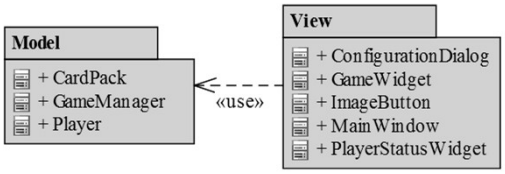
Esettanulmányok
Memory játék

Szerkezeti tervezés:

- a játékfelületet megjelenítő vezérlőt (`GameWidget`), amely tartalmazza a játékmezővel kapcsolatos tevékenységeket
- ehhez segédosztályként
  - a felhasználói információkat kiíró vezérlőt (`PlayerStatusWidget`, ezt előléptetett vezérlővel állítjuk be a felülettervezőben)
  - a képet megjeleníteni tudó egyedi gombot (`ImageButton`)
- A nézet a modell publikus műveleteit hívja, és eseményeket is kaphat tőle

Esettanulmányok
Memory játék

Szerkezeti tervezés (csomagok):

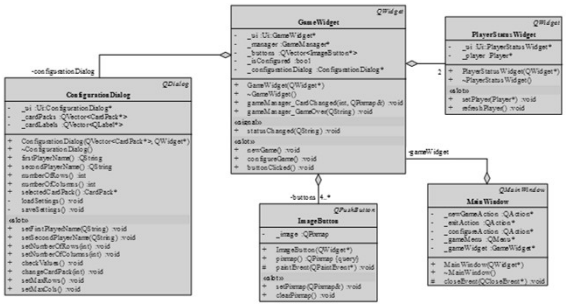


Esettanulmányok
Memory játék

Szerkezeti tervezés (modell):

Esettanulmányok
Memory játék

Szerkezeti tervezés (nézet):

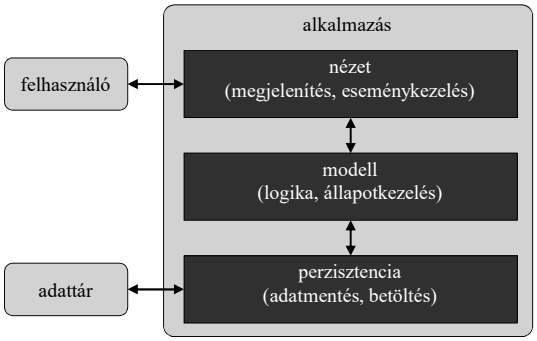


Objektumorientált tervezés: architektúra
A háromrétegű architektúra

- Sok alkalmazásban megjelenik a hosszú távú adattárolás, a *perzisztencia* (*persistency*) feladatköre
  - megadja az adattárolás helyét (fájl, adatbázis, hálózati szerver, …), és formáját (szöveg, XML, SQL, …)
  - általában független a nézettől és a modelltől, ezért külön csomagként kezelendő
- A *háromrétegű* (*three-tier*) architektúra a leggyakrabban alkalmazott szerkezeti felépítés, amelyben elkülönül:
  - a *nézet* (*presentation/view tier*, *presentation layer*)
  - a *modell* (*logic/application tier*, *business logic layer*)
  - a *perzisztencia*, vagy *adatelérés* (*data tier*, *data access layer*, *persistence layer*)

Objektumorientált tervezés: architektúra

A háromrétegű architektúra

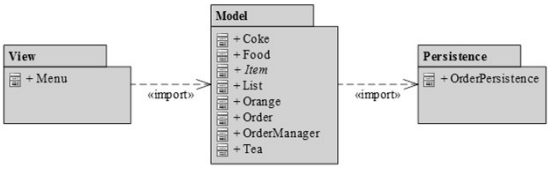


Esettanulmányok

Marika néni kávézója

Szerkezeti tervezés:

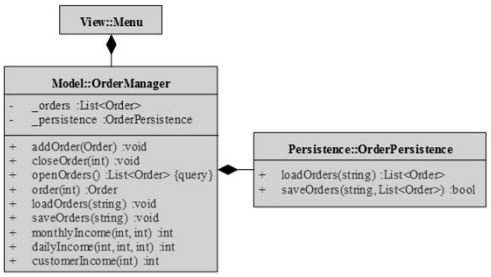
- Az alkalmazást háromrétegű architektúrában valósítjuk meg, a perzisztencia (`OrderPersistence`) felel az adatok mentéséért, betöltéséért



Esettanulmányok

Marika néni kávézója

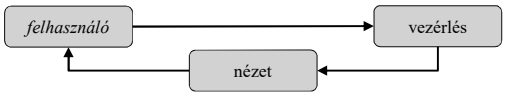
Szerkezeti tervezés:



Objektumorientált tervezés: architektúra

Az MVC architektúra

- Asztali környezetben a felhasználó a nézettel teremt kapcsolatot, amely biztosítja a megfelelő utasítás végrehajtását
- Webes környezetben a felhasználó az adott erőforrással teremt kapcsolatot, amit elsősorban az útvonala (URL) határoz meg
  - vagyis a felhasználó közvetlenül a vezérlést veszi igénybe
  - a vezérlésre az alkalmazásnak egy (új) nézettel kell válaszolnia, ami az adott erőforráshoz tartozik



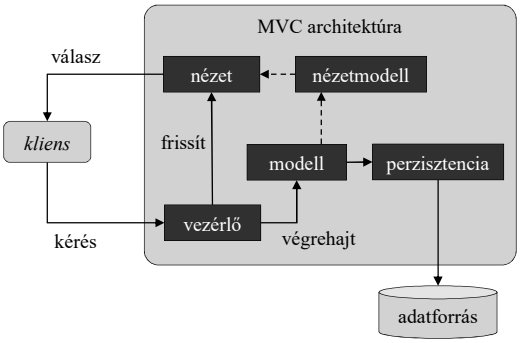
Objektumorientált tervezés: architektúra

Az MVC architektúra

- A *modell/nézet/vezérlő* (*Model-View-Controller*, *MVC*) architektúra egy többrétegű felépítést definiál, amely jól illeszkedik a webes környezethez
  - a *vezérlő* a kérések kiszolgálója, amely biztosítja a nézetet a kérés eredménye alapján
  - a *nézet* a felület (jórészt deklaratív) definíciója, nem tartalmaz háttérkódot, csupán az adatokat kéri a modelltől
  - a *modell* a logikai funkciók végrehajtása (üzleti logika)
  - a *nézetmodell* egy átjáró, amely az adatokat a nézet számára megfelelő módon prezentálja
  - a *perzisztencia* felel az adatelérésért

Objektumorientált tervezés: architektúra

Az MVC architektúra



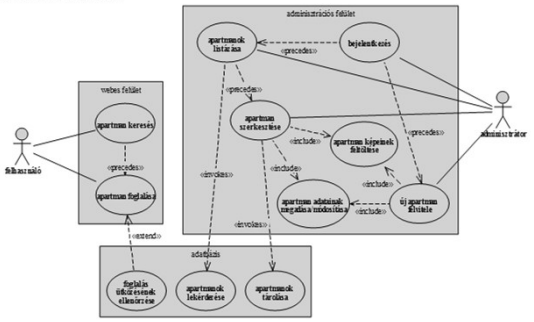
Esettanulmányok
Utazási ügynökség

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

Használati esetek:



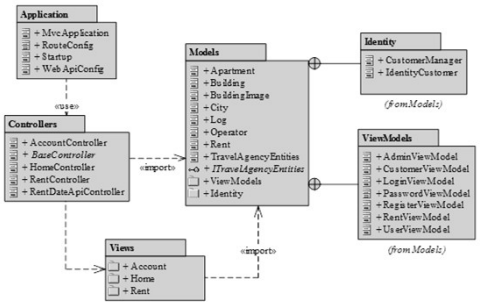
Esettanulmányok
Utazási ügynökség

Szerkezeti tervezés:

- A webes felületet MVC architektúrában valósítjuk meg
  - a felületet egy vezérlő (`HomeController`) irányítja, amely három akciót definiál: minden listázása (`Index`), egy város épületeinek listázása (`List`), egy épület részleteinek lekérése (`Details`)
  - egy vezérlő a foglalásokat felügyeli (`RentController`), két nézettel
  - egy vezérlőben (`AccountController`) kezeljük a regisztráció (`Register`), bejelentkezés (`Login`) és kijelentkezés (`Logout`) funkciókat, amelyekhez két új nézetet készítünk
  - …

Esettanulmányok
Utazási ügynökség

Szerkezeti tervezés:



Esettanulmányok
Utazási ügynökség

Szerkezeti tervezés: