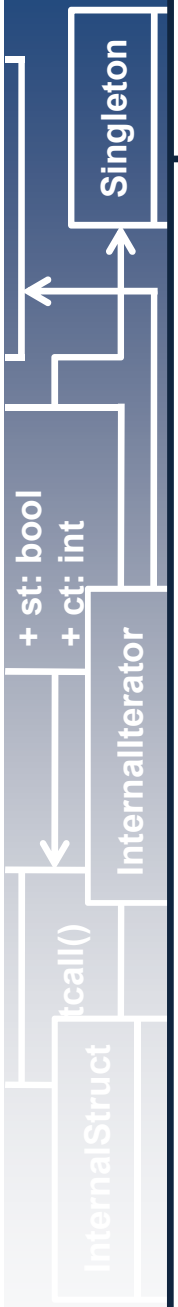
ELTE-IK
Programozási technológia 1.

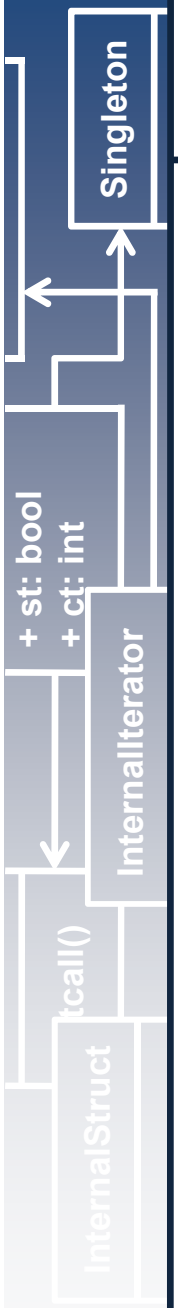
3. gyakorlat:
Objektumorientált tervezés, az UML nyelv

© 2008.09.24. Giachetta Roberto
groberto@inf.elte.hu
http://people.inf.elte.hu/groberto



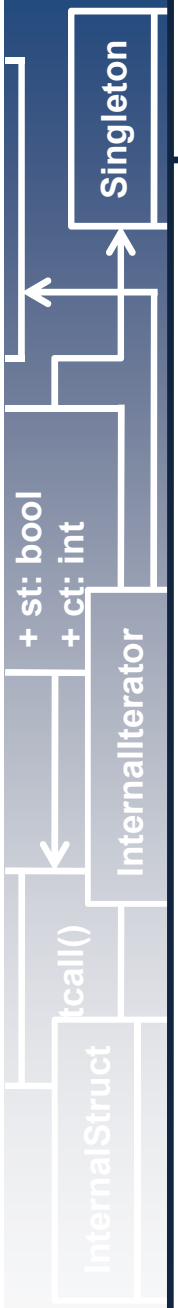
Objektumorientált programozás

- Az objektumorientált programozásban egymással kommunikáló **objektum**ok összessége valósítja meg a feladatot
  - minden objektum képes önálló működésre, és megvan a saját feladatköre
  - az objektumok egymásnak üzeneteket küldenek, amelynek hatására az objektumok valamilyen működést végeznek, objektumok kapcsolatát **reláció**nak nevezzük
  - az objektumok állapottal rendelkeznek, ahol az állapot a benne tárolt összes érték összessége, és ami üzenet vagy a működés hatására megváltozhat
  - minden objektum önállóan azonosítható, és két objektum még akkor is különbözik egymástól, ha teljesen azonos állapotban vannak



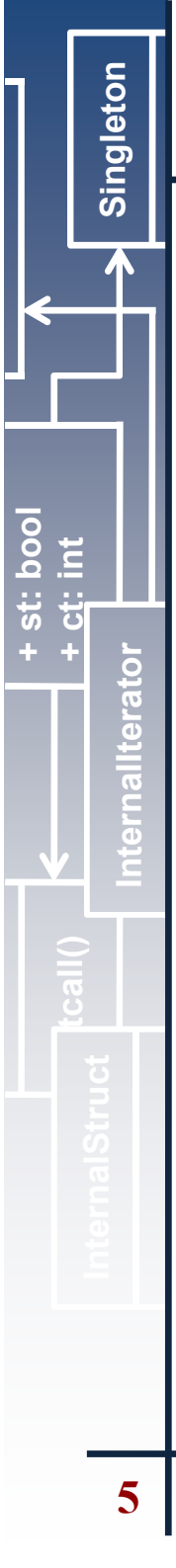
Objektumorientált programozás

- Az objektumoknak típusa van, amit **osztály**nak hívunk, ez szabályozza milyen műveletekre képes az egyed és milyen jellemzőkkel ruházható fel (tehát magában foglalja a **metódus**okat és **attribútum**okat, ez az **enkapszuláció**)
  - az, hogy az osztályt mivel ruházzuk fel attól függ, milyen **absztrakció**t választunk, hiszen nagyobb absztrakciós szintnél elhanyagoljuk a lényegtelenebb tulajdonságokat, mindig a feladat határozza meg, mennyire kell precízen modelleznünk a feladatot
  - szabályozhatjuk, hogy az osztály mely tulajdonságai mely további objektumok számára láthatóak, a metódusok pontos működését persze elrejtjük a külvilágtól (láthatóság szabályozás), de biztosítanunk kell megfelelő publikus felületet (interfészt) az üzenetátadáshoz



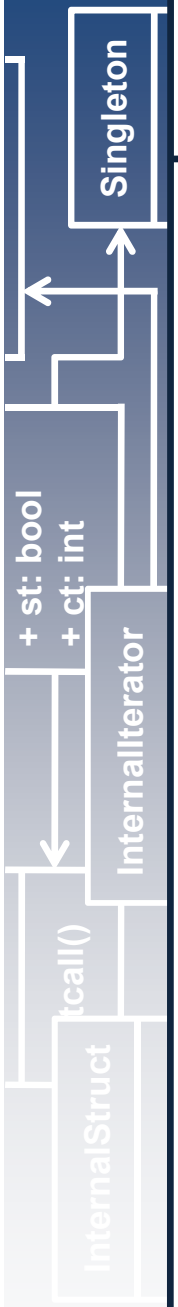
Objektumorientált programozás

- Az osztályok is kapcsolatban állhatnak egymással, ahol osztályból **származtatunk** további osztályokat, és a speciálisabb osztályok átveszik az általánosabbak tulajdonságait, ezt **öröklődés**nek nevezzük
  - általában egyszeres, de lehet többszörös is
  - megadható minden objektumnál, hogy melyik osztályból lett példányosítva, de ugyanakkor az az objektum az osztályának összes ősének is példánya, és adott helyzetben adott osztályúnak tekinthető, ez a **polimorfizmus**



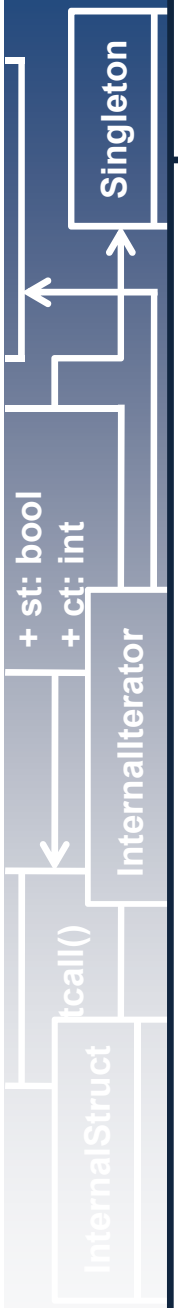
Objektumorientált modellezés

- Az objektumorientált programozásnak sokáig nem volt egységes modellező felülete, azaz nem volt olyan eszköz, amely objektumorientált programok tervezését nyelvtől függetlenül meg tudta volna adni
- 1989-ben megszületett az Object Management Group (OMG), amelynek feladata az objektumorientált tervezés szabványosítása
- 1997-re megszületett a **Unified Modelling Language** (**UML**), mely egy vizuális programozási nyelv szoftverrendszerek absztrakt modelljeinek megalkotására
- Ebből fejlődött ki 2001-re a Modell Driven Architecture (MDA), amely egy teljes tervezési és platform független modellezési eljárást foglal magába, amely üzleti modelleket is képes előállítani



Objektumorientált modellezés

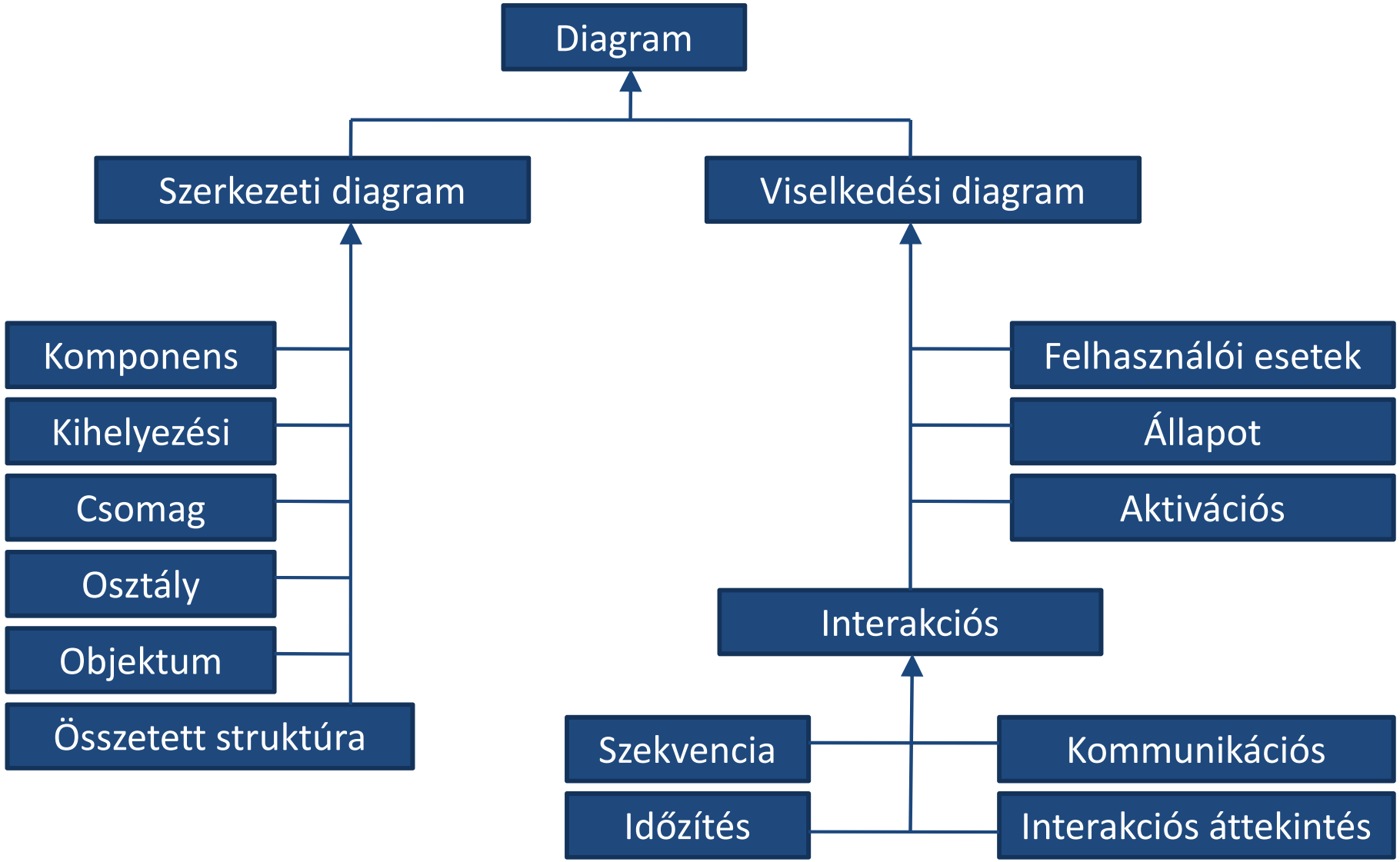
- Az UML segítségével szabványos módon lehet rendszerek terveit elkészíteni
  - alkalmas üzleti folyamatok és programfunkciók, és adatbázis-sémák leírására
  - a modellek automatikusan kódba fejthetőek, tehát tetszőleges objektumorientált nyelvre átültethetőek
  - a nyelv kiterjeszthető, és lehetőséget ad a személyesítésre
  - a nyelv leginkább diagramok keretében mutatkozik meg, noha a nyelv nem a diagramokat magukat adja meg, hanem a diagramok által reprezentált modell specifikációját
- 2003-ban készült el az UML 2.0-s specifikációja, amely 13 diagramtípust vezet be



Objektumorientált modellezés

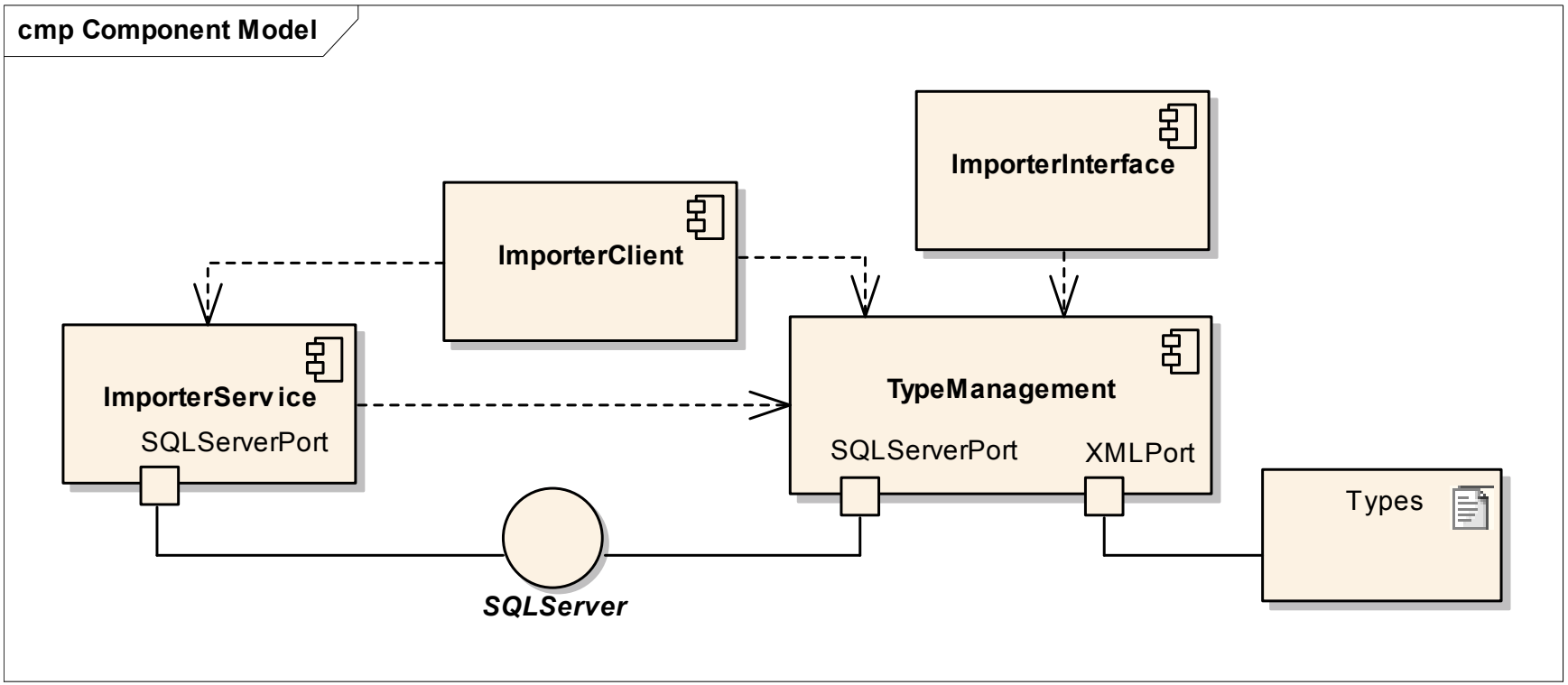
- Az UML a szoftverrendszert a következő szempontok szerint tudja jellemezni:
  - funkcionális modell: a szoftver funkcionális követelményeit adja meg és a felhasználóval való interaktivitást, pl.: felhasználói esetek diagramja, kihelyezési diagram
  - szerkezeti modell: a program felépítését adja meg, milyen osztályok, objektumok, relációk alkotják a programot, pl.: osztálydiagram, objektumdiagram
  - dinamikus modell: a program működésének lefolyását, az objektumok együttműködésének módját ábrázolja, pl.: állapotdiagram, szekvenciadiagram

Az UML diagramjai



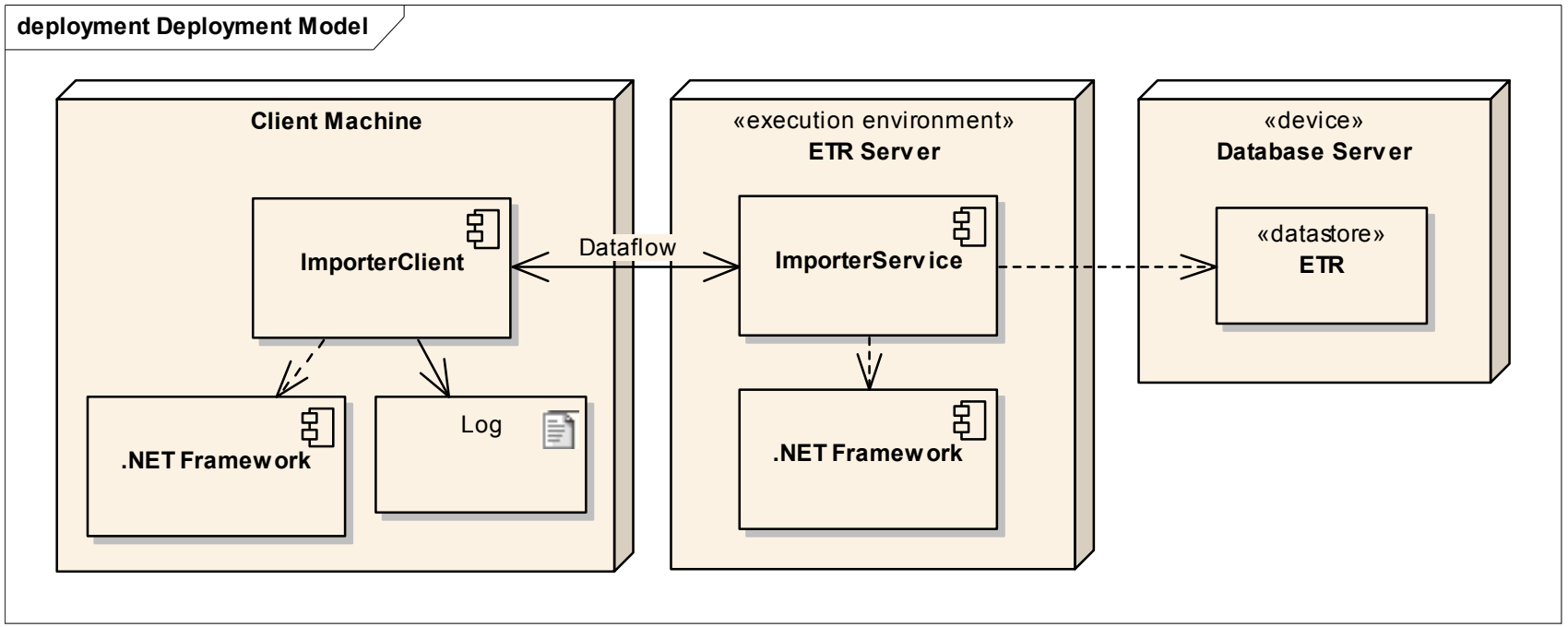
Komponensdiagram

- Amennyiben a rendszer több komponensből (programegységből) áll, akkor azokat függőségeikkel együtt ábrázolja
- A komponensek interfészeit portokkal ábrázolja, ezek adják meg a kommunikációs felületet



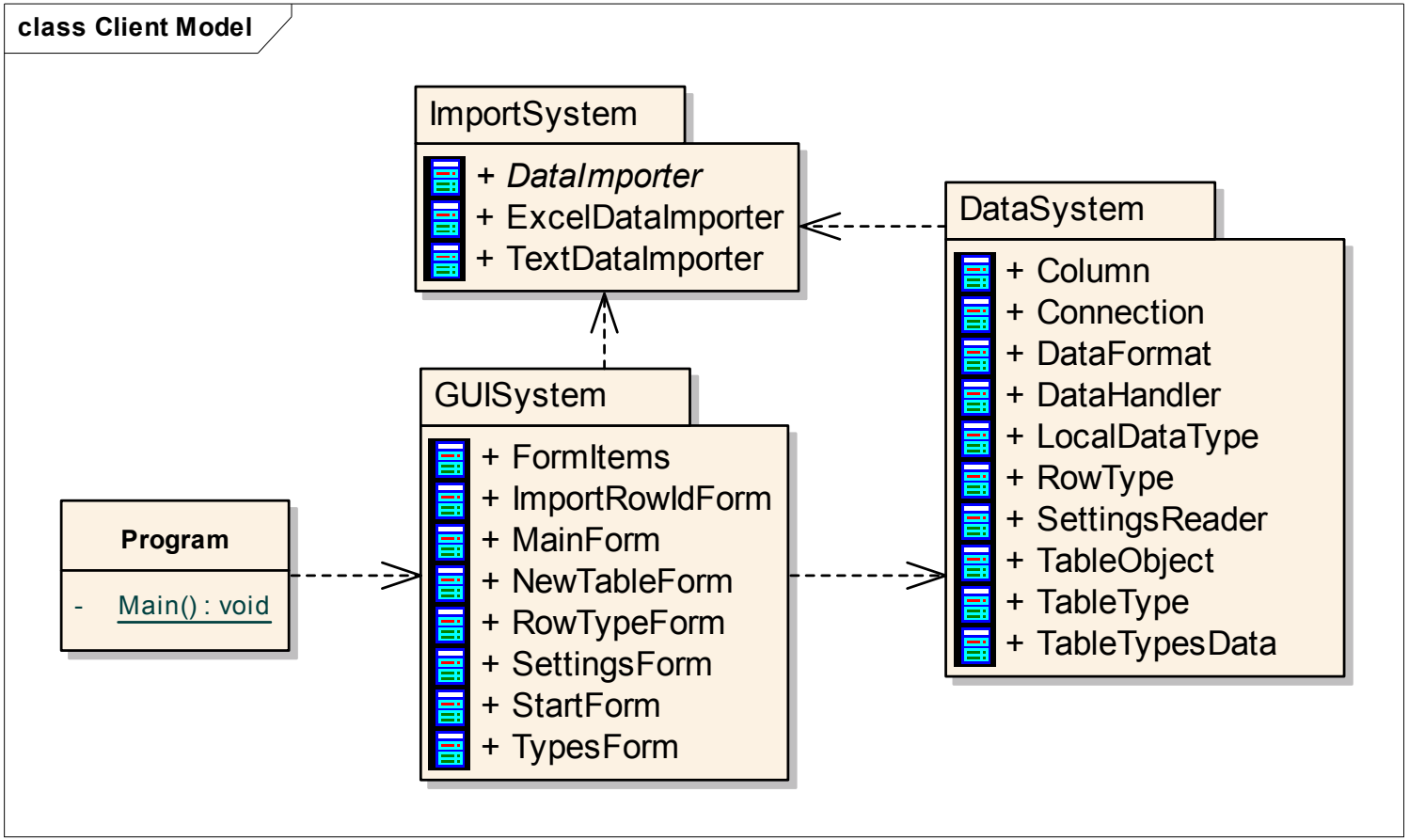
Kihelyezési diagram

- Megadja a szoftverrendszer kihelyezésének környezetét, illetve az egyes komponensek elhelyezkedését a környezetben
- Láthatóak a szoftver működéséhez szükséges előfeltételek, illetve a kapcsolódó adattárak



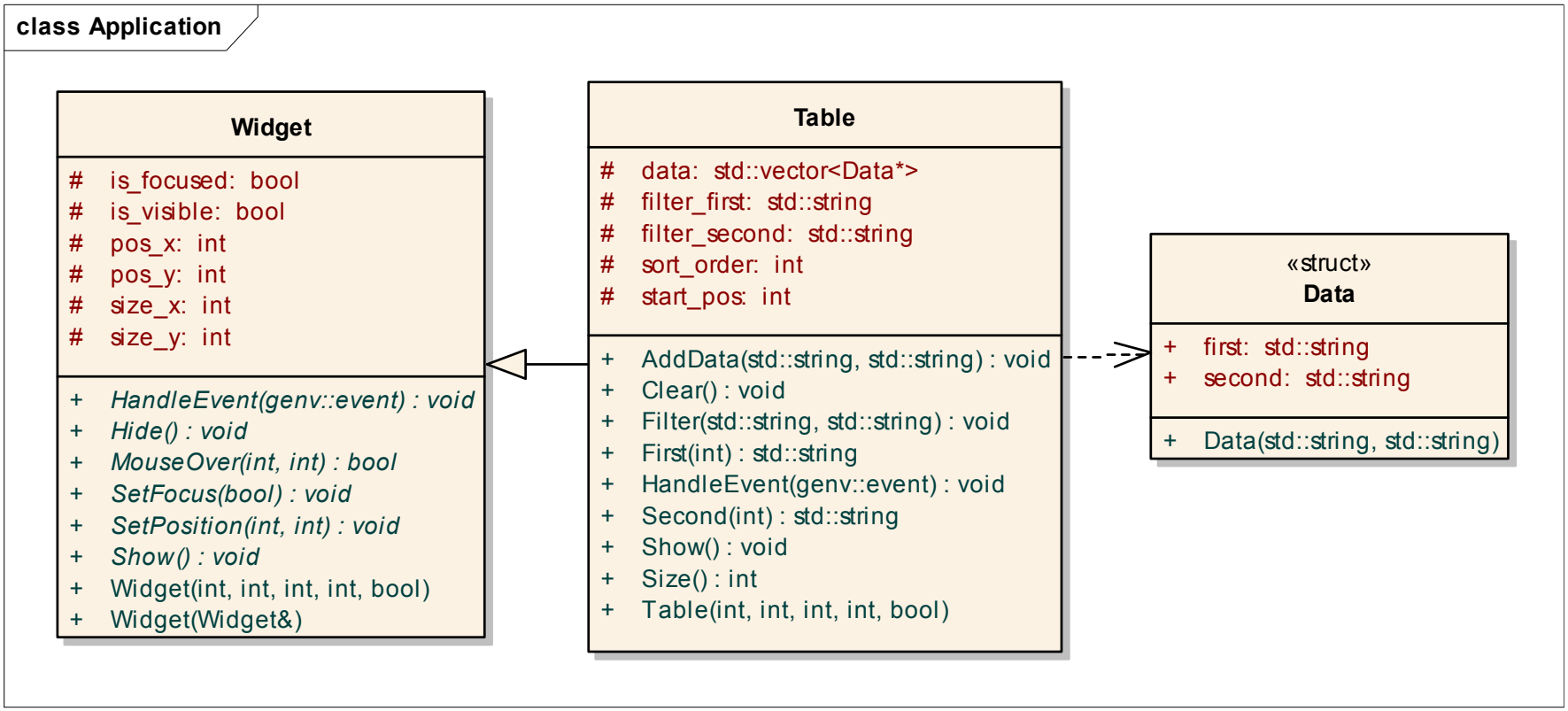
Csomagdiagram

- Az egyes komponensek logikai felépítését taglalja, azaz milyen csoportokat alakítanak az osztályok
- Általában a csomagok a program névtereit adják meg



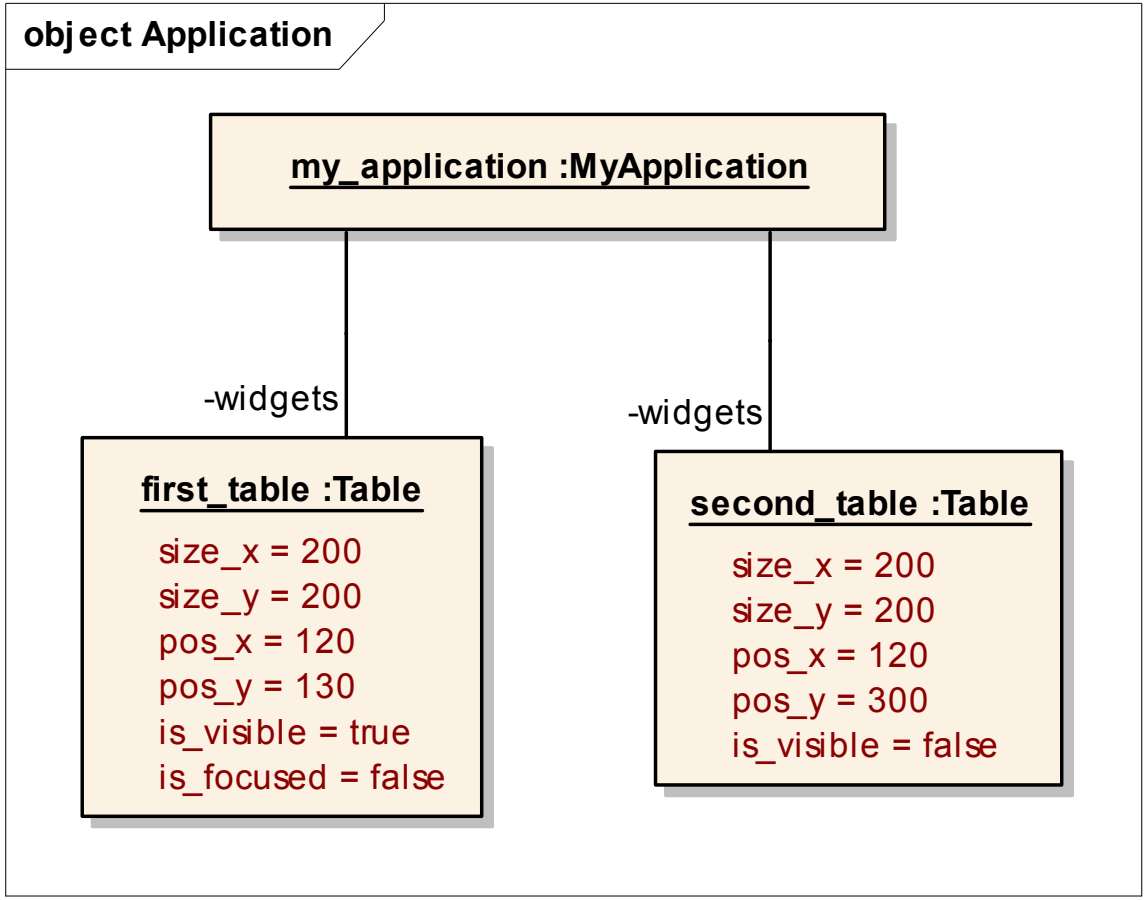
Osztálydiagram

- Megadja a rendszer osztályait és azok kapcsolatát (általában egy komponensen, vagy csomagon belül)
- Általában ábrázolja az attribútumokat és metódusokat láthatóságukkal és paraméterlistával



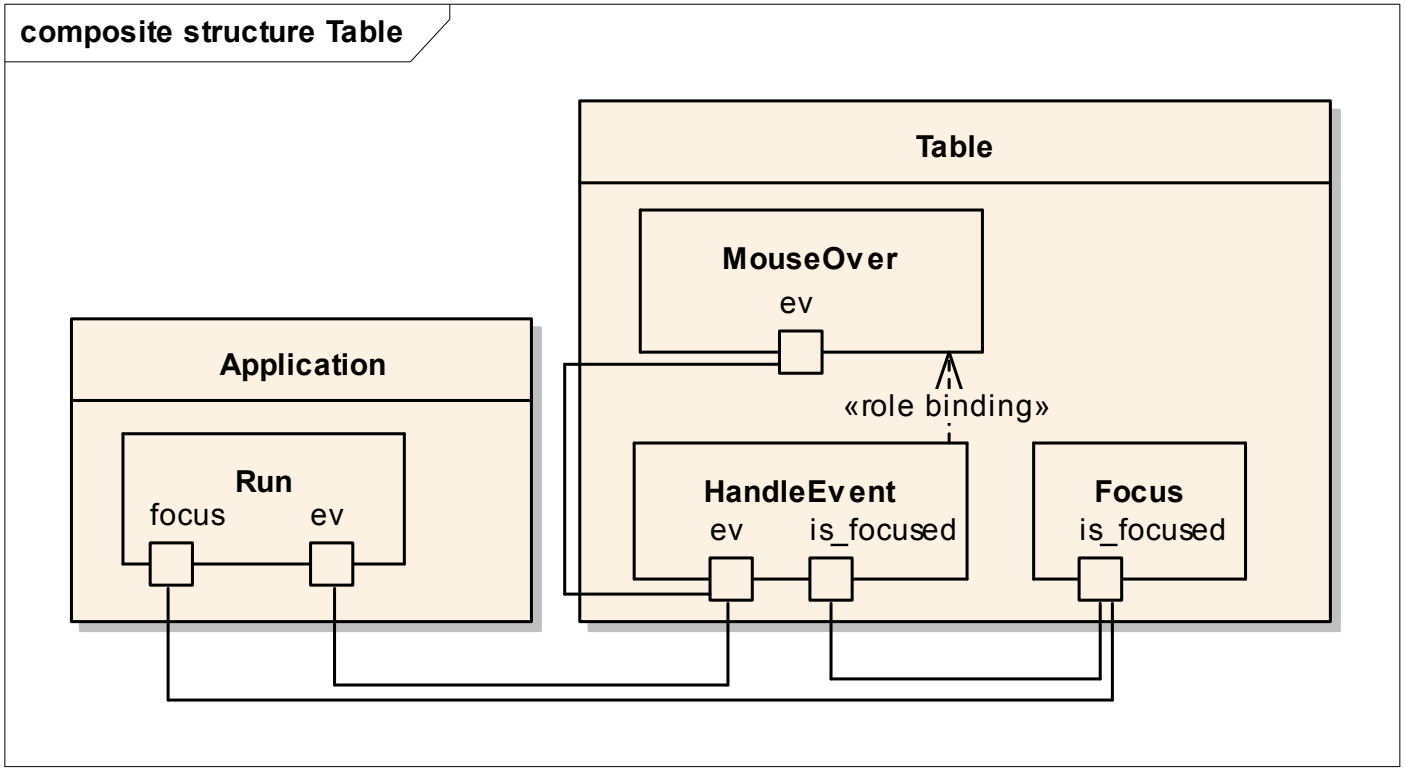
Objektumdiagram

- A programban szereplő objektumokat és relációikat ábrázolja
- Minden objektumot egy adott állapotban ad meg, azaz az attribútumok konkrét értékeket kapnak



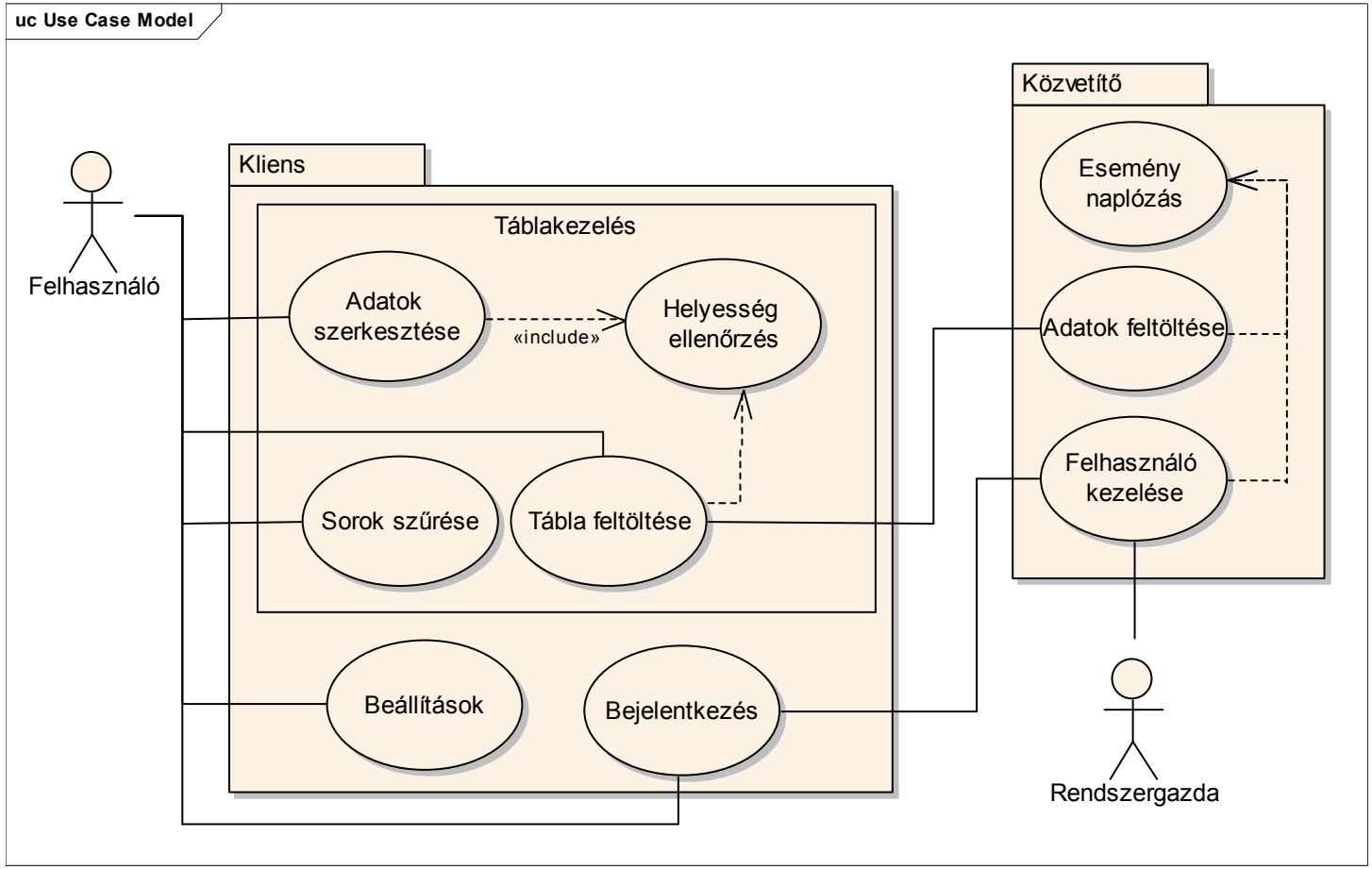
Összetett struktúra diagram

- Egy osztály belső szerkezetét adja meg, és hogy az osztály milyen együttműködéseket végezhet
- Megjelennek az osztály metódusai, mint a kommunikáció részesei, illetve paraméterei és attribútumai, mint az adatok forrásai



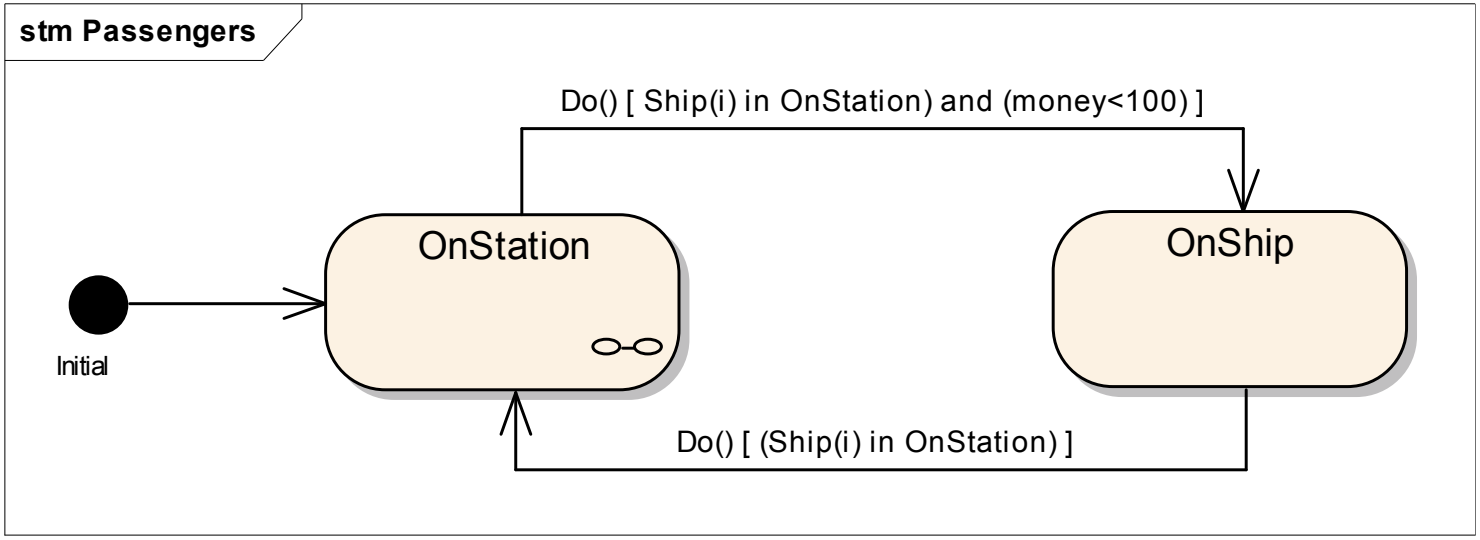
Felhasználói esetek diagramja

- Megadja a szoftverrendszer és a felhasználó interakciójának lehetőségeit, a külső programfunkciókat programegységekre, illetve csomagokra csoportosítva



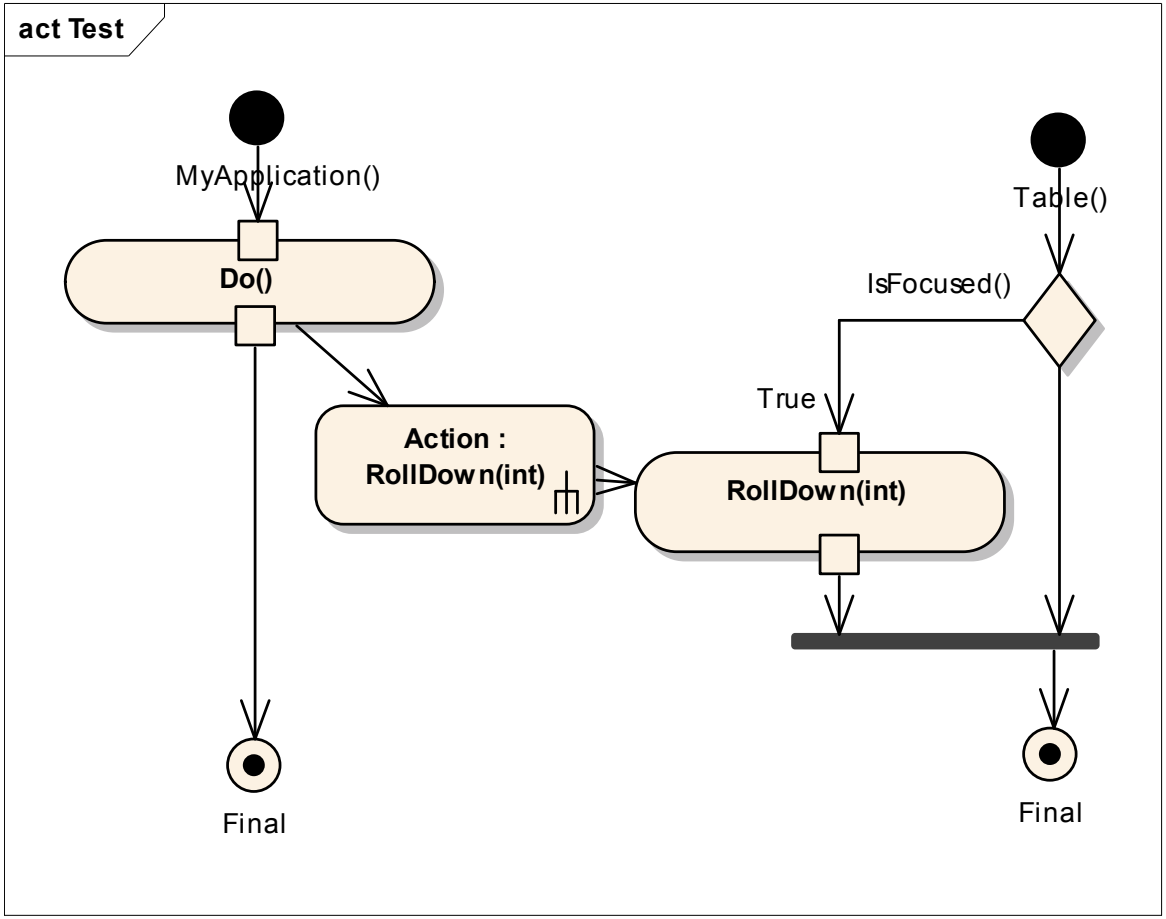
Állapotdiagram

- Reprezentálja az egyes osztályok állapotait, illetve állapotváltozásait a futás során beleértve a létrehozást és terminálást is
- Részletesen megadható, milyen események milyen feltételek mellett vezetnek az állapotváltozáshoz



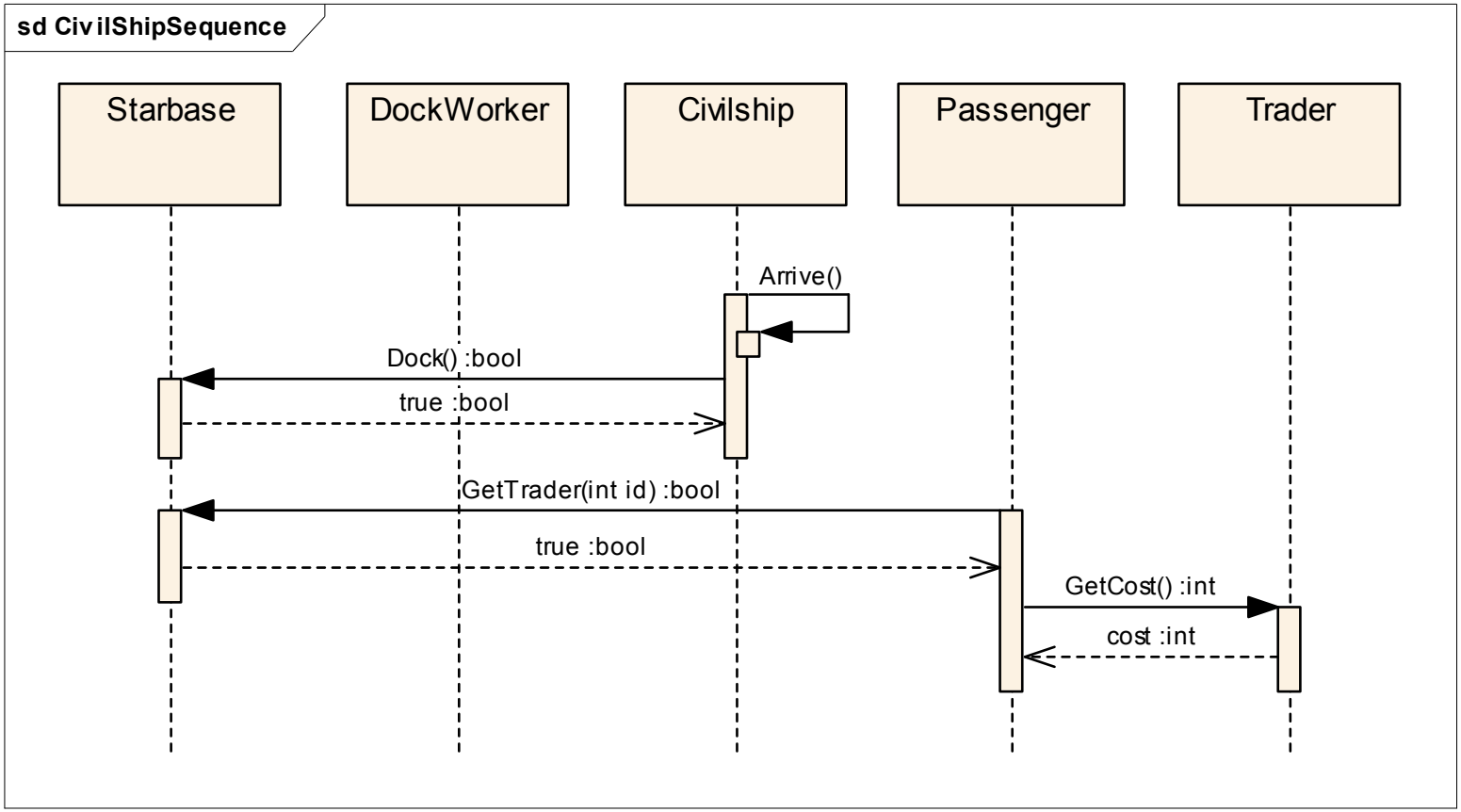
Aktivációs

- A programfolyamatban keletkező aktivációkat kezeli, azaz, hogy milyen tevékenységeket hajt végre egy adott objektum a futás során



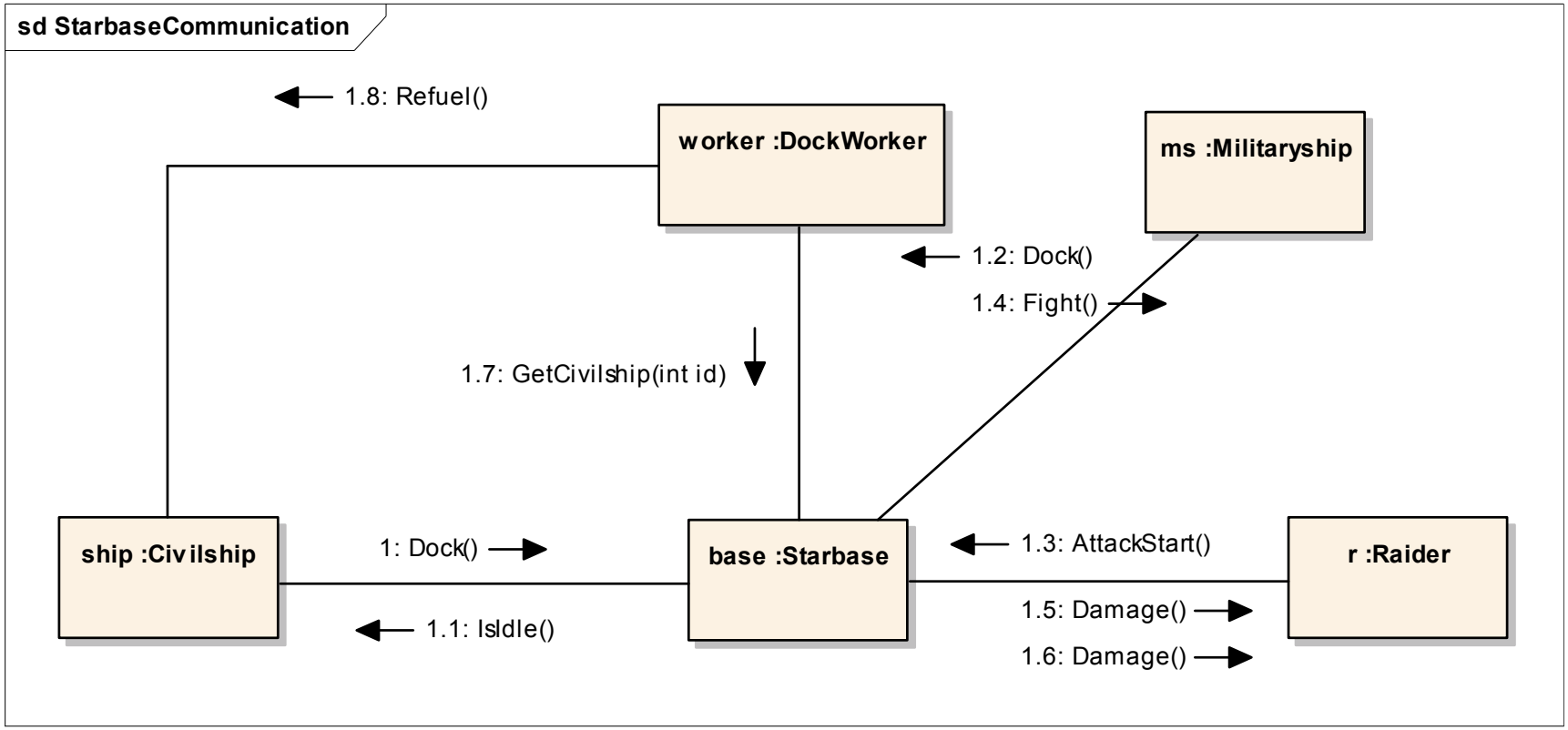
Szekvencia diagram

- A program futása során időben az egyes objektumok közötti életfolyamot mutatja az üzenetátadásokkal
- Megadja, mely objektumoknál van a vezérlés az adott időpillanatban



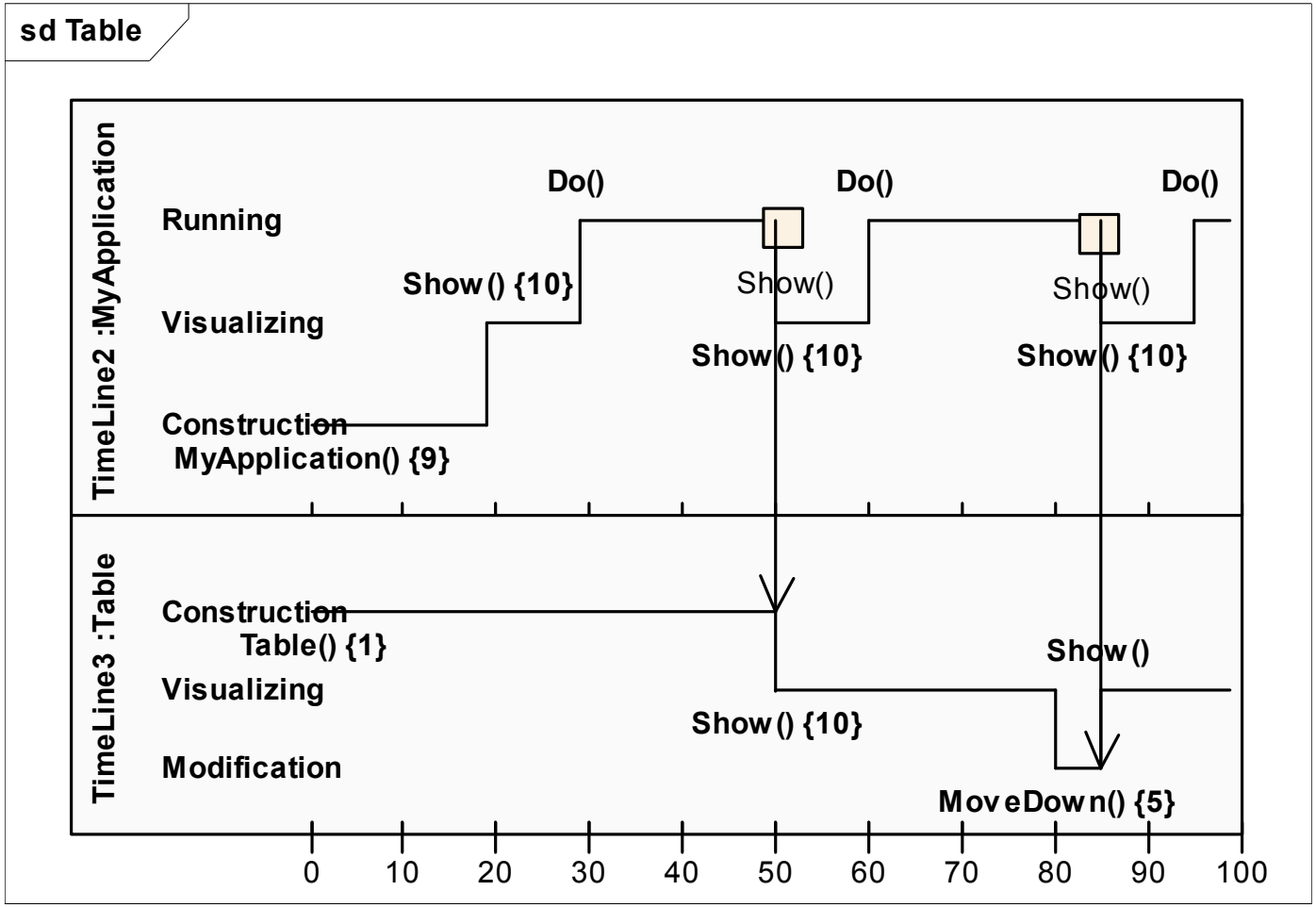
Kommunikációs diagram

- Az objektumok közötti üzenetátadást és azok sorrendjét ábrázolja
- Az üzenetátadáson kívül megegyezik az objektum diagrammal



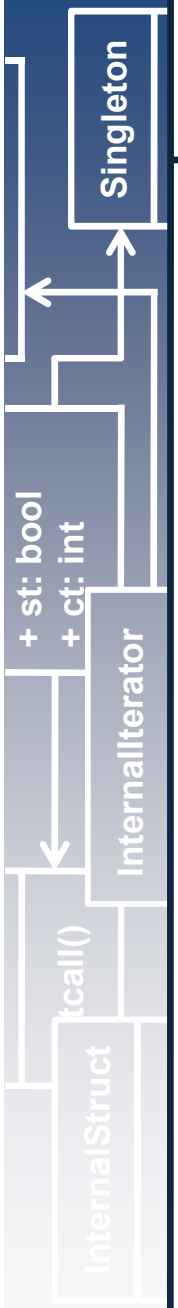
Időzítés diagram

- Időben (konkrétan, vagy viszonyítási alapon) adja meg az objektumok állapotváltozásait, illetve az üzenetek lefolyását



Interakciós áttekintés diagram

- Olyan aktivációs diagram, ahol minden aktivációt egy szekvencia ad meg, azaz minden aktivációs pont egy szekvenciadiagramot tartalmaz



UML diagramok használata

A szoftverfejlesztés különböző szakaszaiban az UML különböző diagramjait kell alkalmaznunk:

1. elemzés: felhasználói esetek, komponens, kihelyezési
2. specifikáció: komponens, kihelyezési
3. tervezés:
   1. statikus tervezés: csomag, osztály, objektum
   2. dinamikus tervezés: állapot, szekvencia, aktivációs, interakciós áttekintési, kommunikációs
4. tesztelés: időzítés

- Persze a későbbi fázisokban a korábban létrehozott diagramok újra alkalmazhatóak, esetlegesen módosíthatóak
- A diagramoknak a fázist követő dokumentációban szerepelnie kell