Programozási Technológia 1.

2. előadás
statikus modell, osztályok

Előadó: Lengyel Zsolt

Tartalom

- Az UML-ről
  - Az UML kialakulása
  - Az UML használata
  - UML diagramok
- A statikus modell
- Objektumelvűség
- Objektumok és osztályaik
- Osztályok és objektumok JAVA-ban

Az UML kialakulása

- 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 létrejött az Object Management Group (OMG), feladata az objektumorientált tervezés szabványosítása
- 1997-re megszületett a *Unified Modelling Language* (*UML*), egy olyan vizuális programozási nyelv, mely szoftverrendszerek absztrakt modelljeinek megalkotására alkalmas
- 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

Az UML használata I/III

- 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 átírhatóak
  - a nyelv kiterjeszthető, és lehetőséget ad a személyesítésre
  - a nyelv leginkább diagramok alakjában jelenik 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

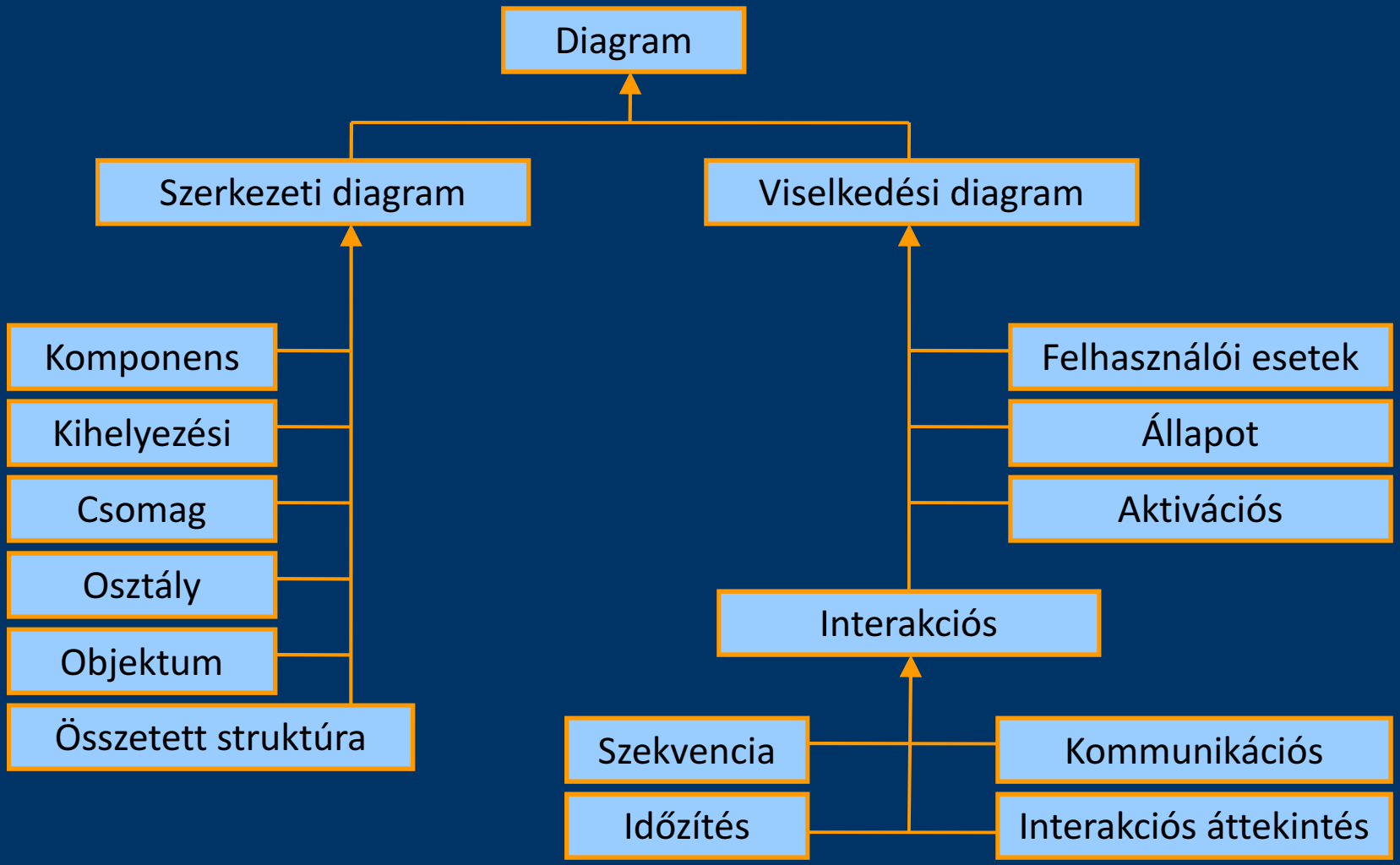
Az UML használata II/III

- Az UML a szoftverrendszert a következő szempontok szerint tudja jellemezni:
  - *funkcionális modell*: a szoftver funkcionális követelményeit és a felhasználóval való interaktivitást adja meg
    - 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 használata III/III

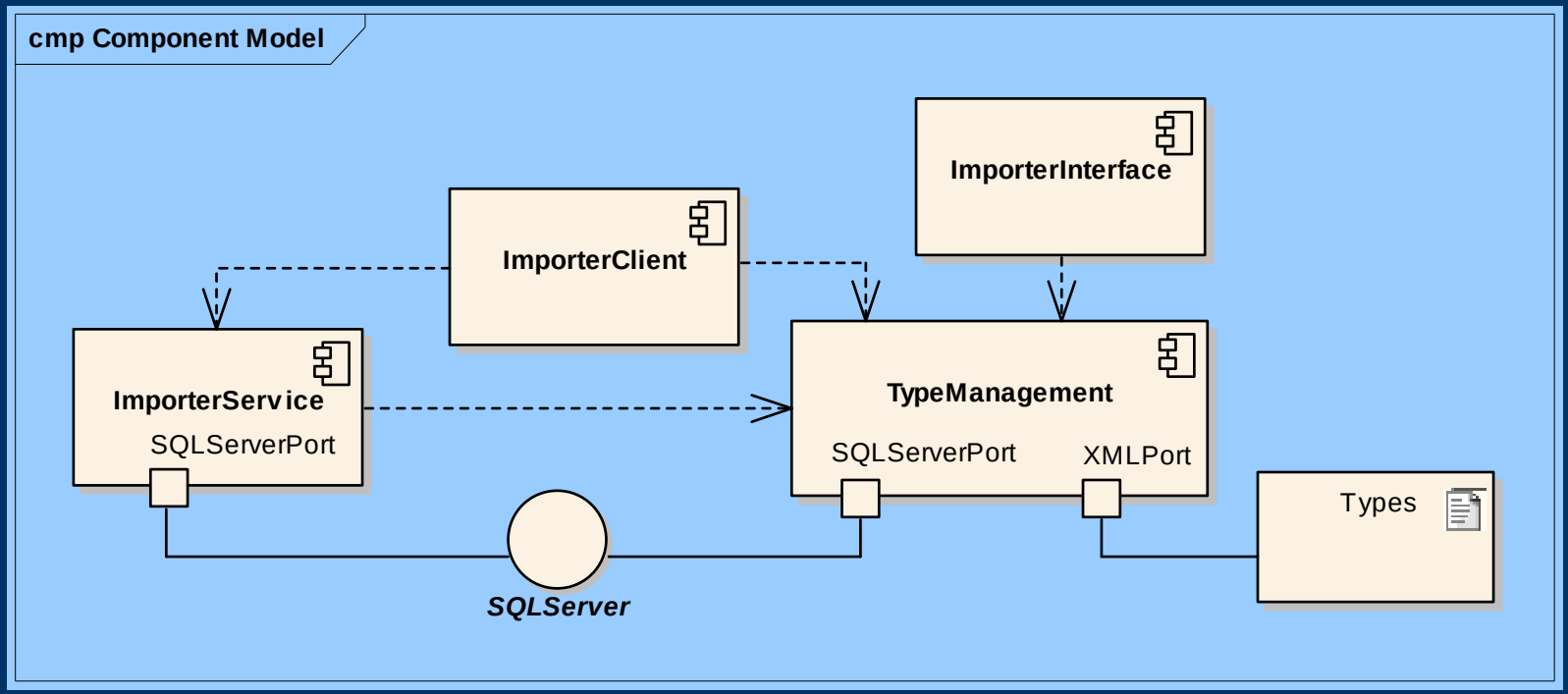
- A szoftverfejlesztés különböző szakaszaiban az UML különböző diagramjait kell alkalmaznunk:
  - elemzés: felhasználói esetek, komponens, kihelyezési
  - specifikáció: komponens, kihelyezési
  - tervezés:
    - statikus tervezés: csomag, osztály, objektum
    - dinamikus tervezés: állapot, szekvencia, aktivációs, interakciós áttekintési, kommunikációs
  - tesztelés: időzítés
- A későbbi fázisokban a korábban létrehozott diagramok újra alkalmazhatóak, esetleg módosíthatóak
- A diagramoknak a fázist követő dokumentációban szerepelniük kell

UML diagramok



UML diagramok
komponensdiagram (component diagram)

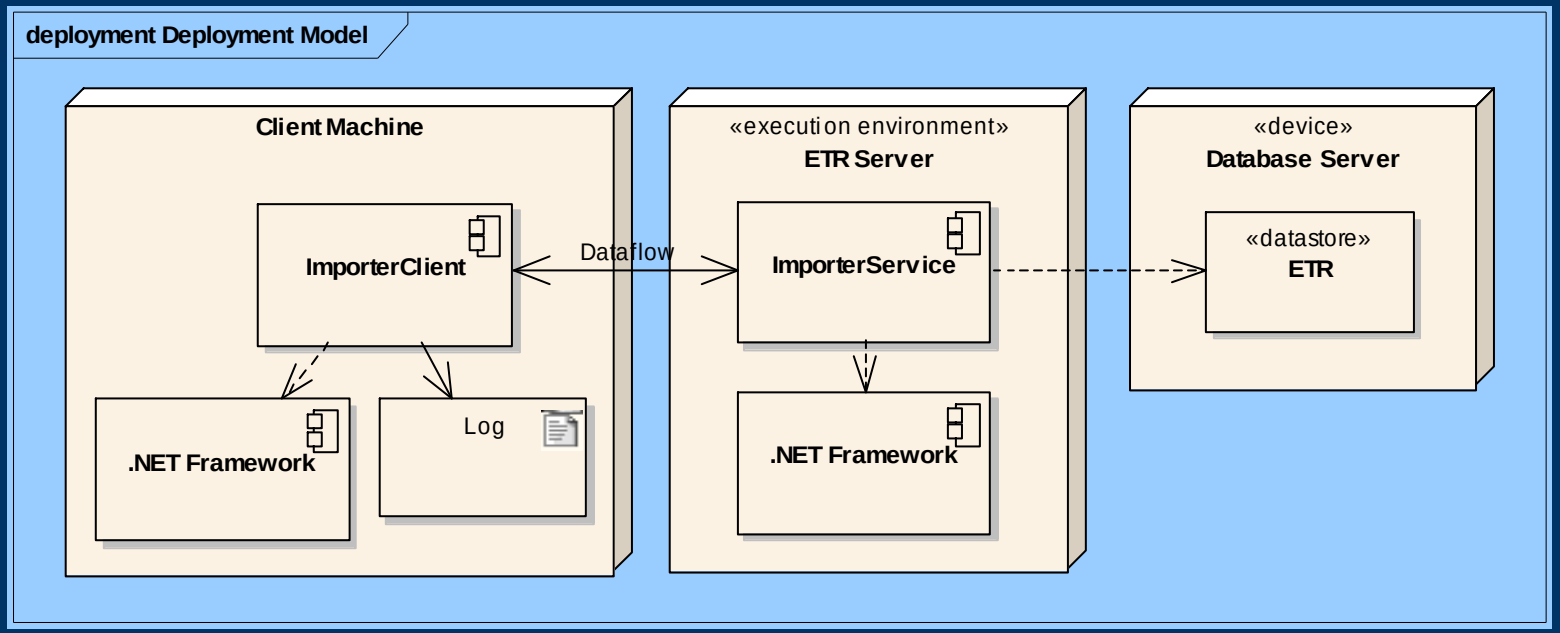
- Amennyiben a rendszer több komponensből áll, akkor azokat függőségeikkel együtt ábrázolja
- A komponensek interfészeit portokkal jeleníti meg, ezek adják meg a kommunikációs felületet



UML diagramok

telepítési diagram (deployment diagram)

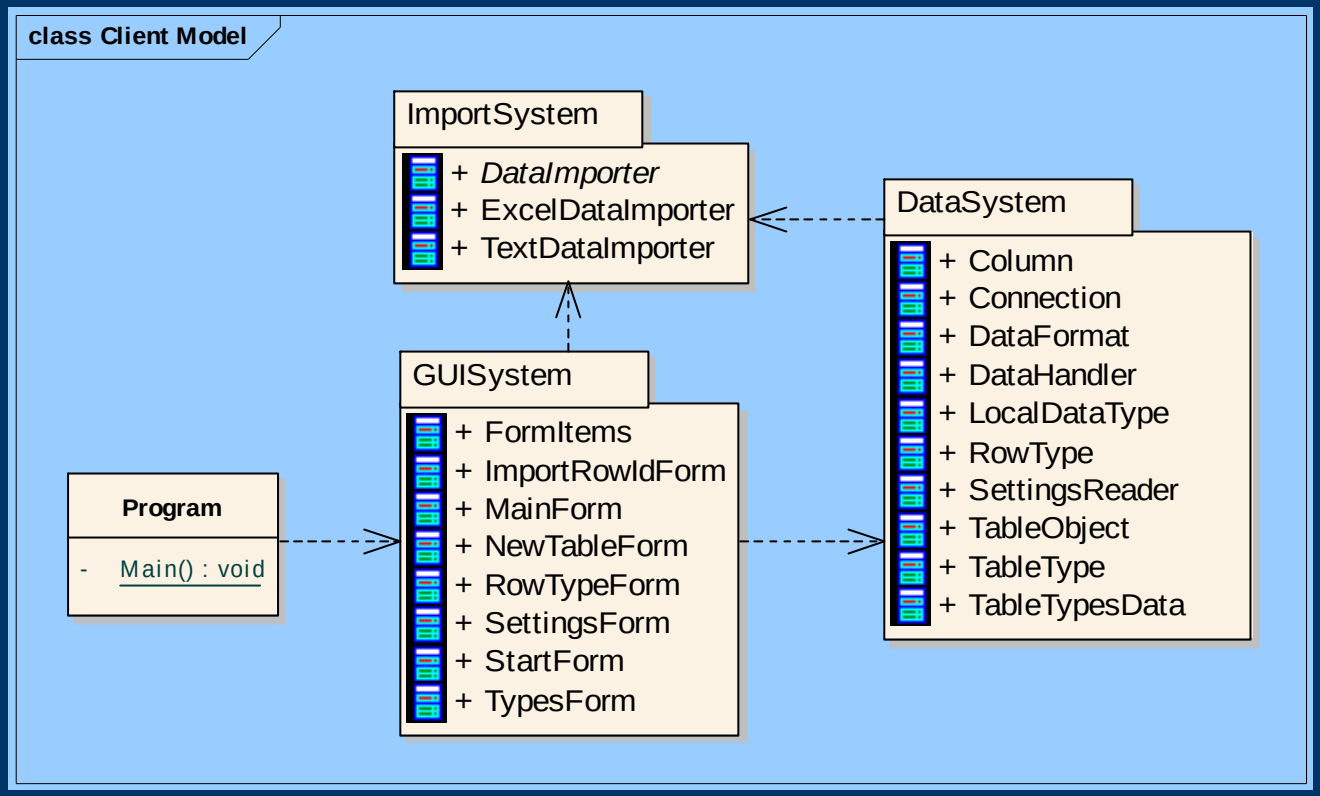
- Megadja a szoftverrendszer telepítésének környezetét, illetve az egyes komponensek elhelyezkedését a környezetben
- Láthatóak a szoftver működéséhez szükséges előfeltételek, illetve a kapcsolódó adattárak



UML diagramok

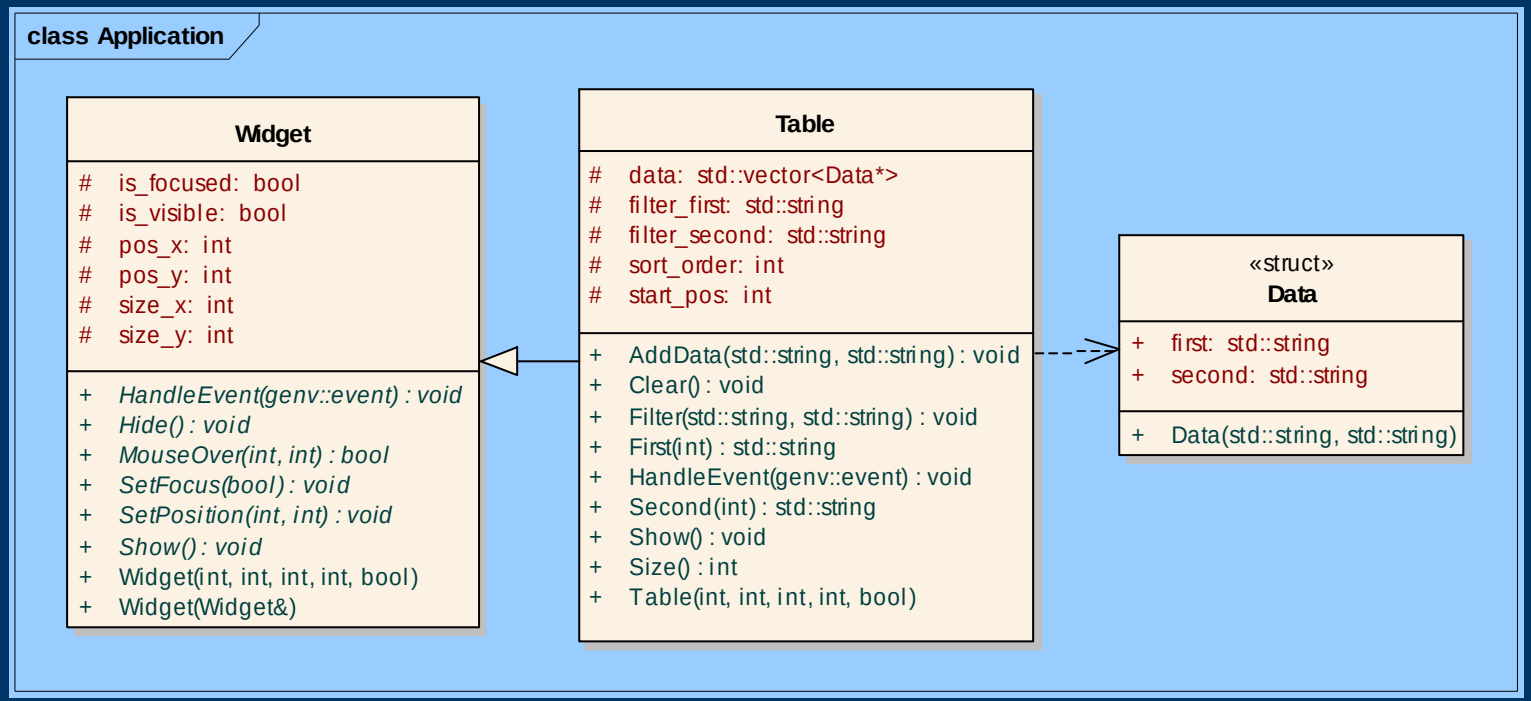
csomagdiagram (package diagram)

- Az egyes komponensek logikai felépítését mutatja, azt, hogy a funkcionalitás szerint összetartozó osztályok milyen csoportokba szervezhetőek
- Általában a csomagok a program névtereit adják meg



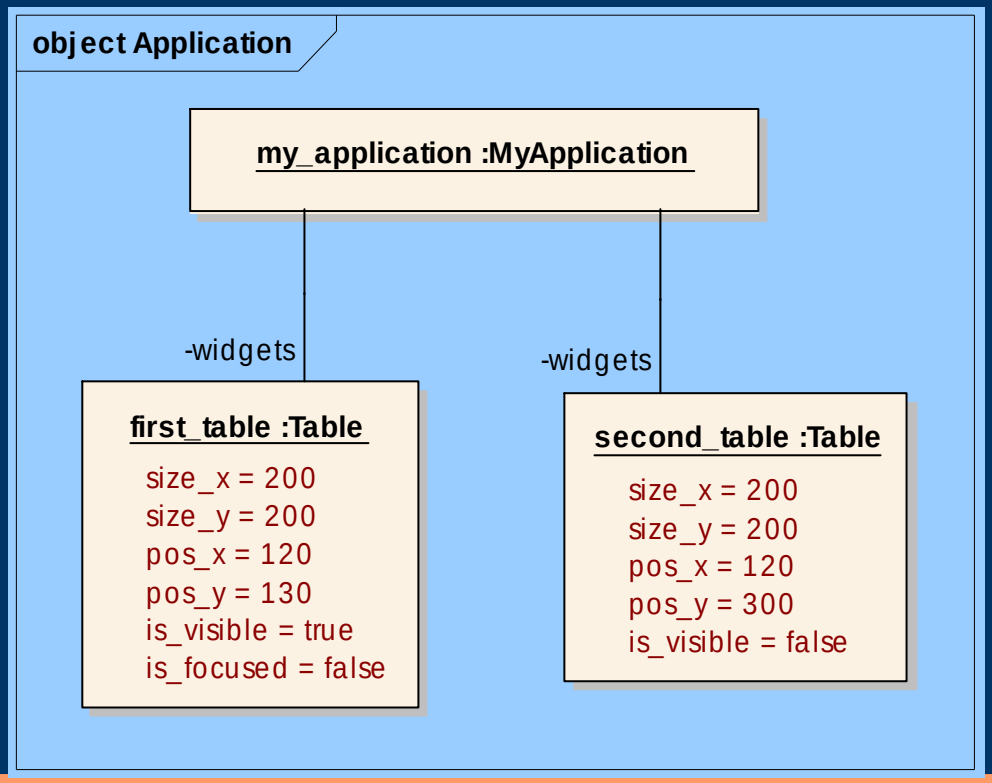
UML diagramok
osztálydiagram (class diagram)

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



UML diagramok
objektumdiagram (object diagram)

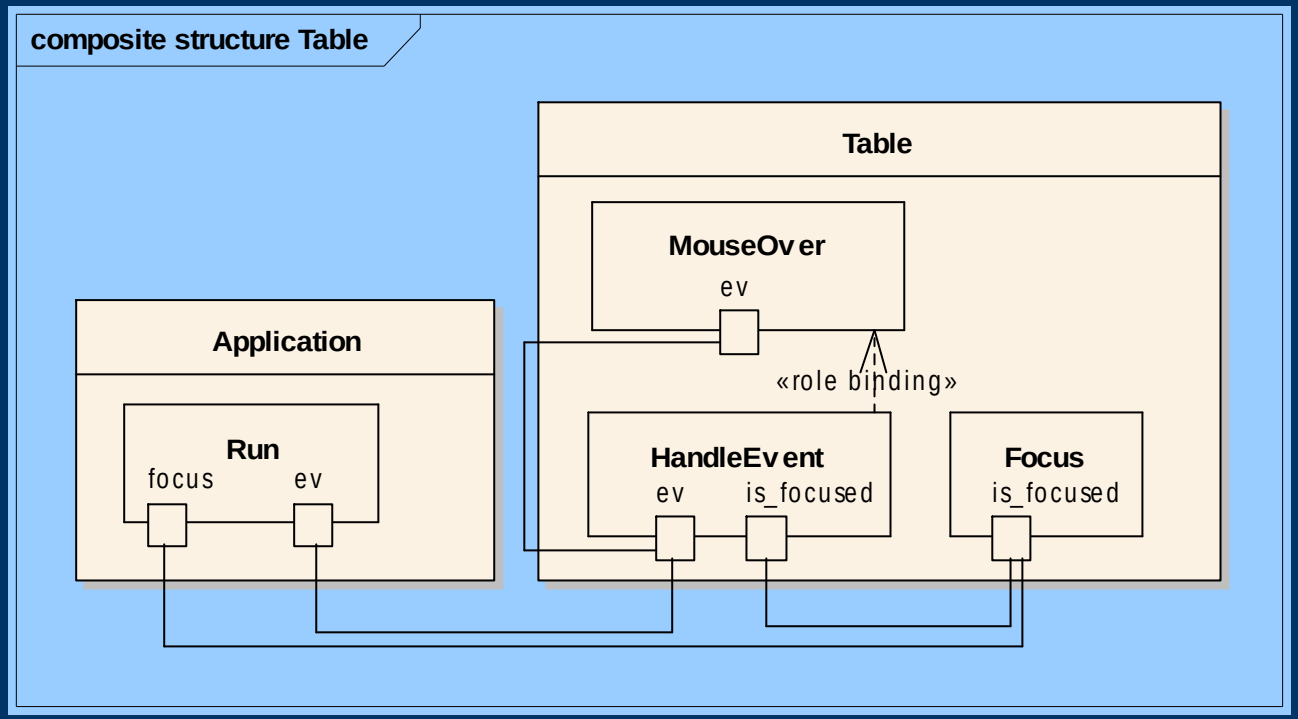
- A programban futása közben lehetségesen szereplő objektumokat és relációikat ábrázolja
- Minden objektumot egy adott állapotában ad meg, azaz az attribútumoknak konkrét értékei láthatóak



UML diagramok

összetett struktúradiagram (composite structure diagram)

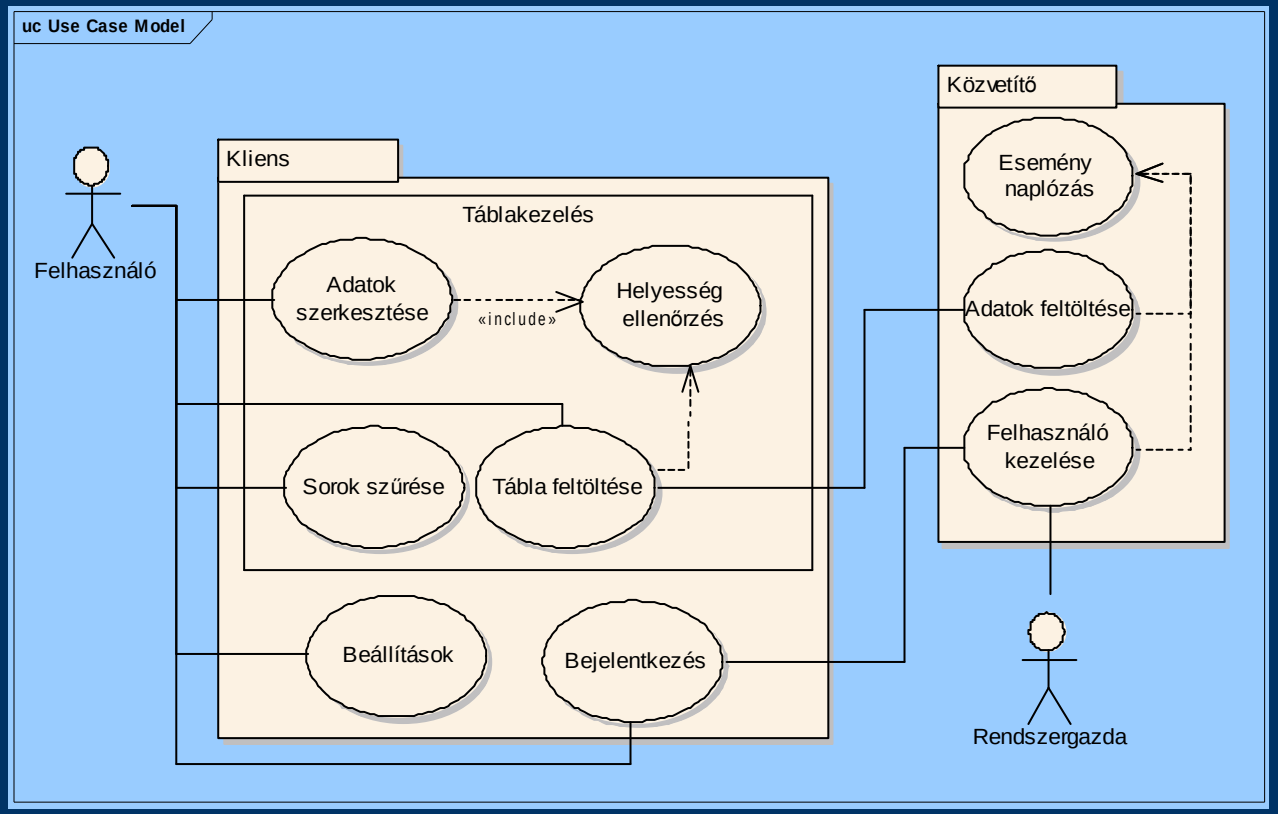
- Egy osztály belső szerkezetét adja meg, és hogy az osztály milyen együttműködéseket végezhet
- Megjelennek az osztály metódusai, mint a kommunikáció részesei, illetve paraméterei és attribútumai



UML diagramok

felhasználói esetek diagramja (use case diagram)

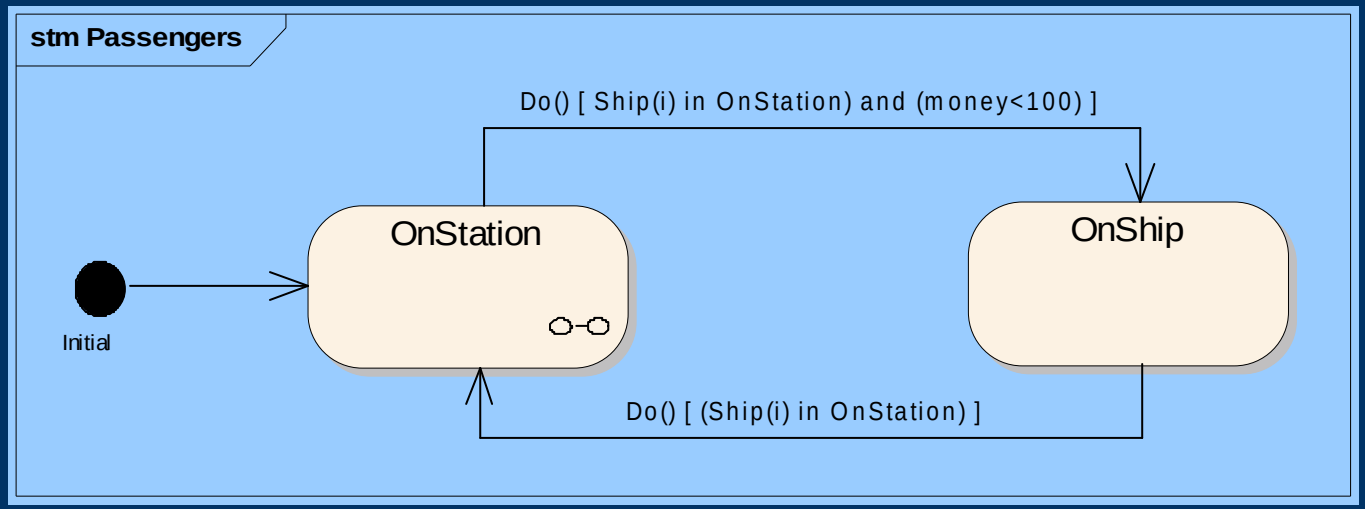
- Megadja a szoftverrendszer és a felhasználó interakciójának lehetőségeit, a külső programfunkciókat programegységekre, illetve csomagokra csoportosítva



UML diagramok

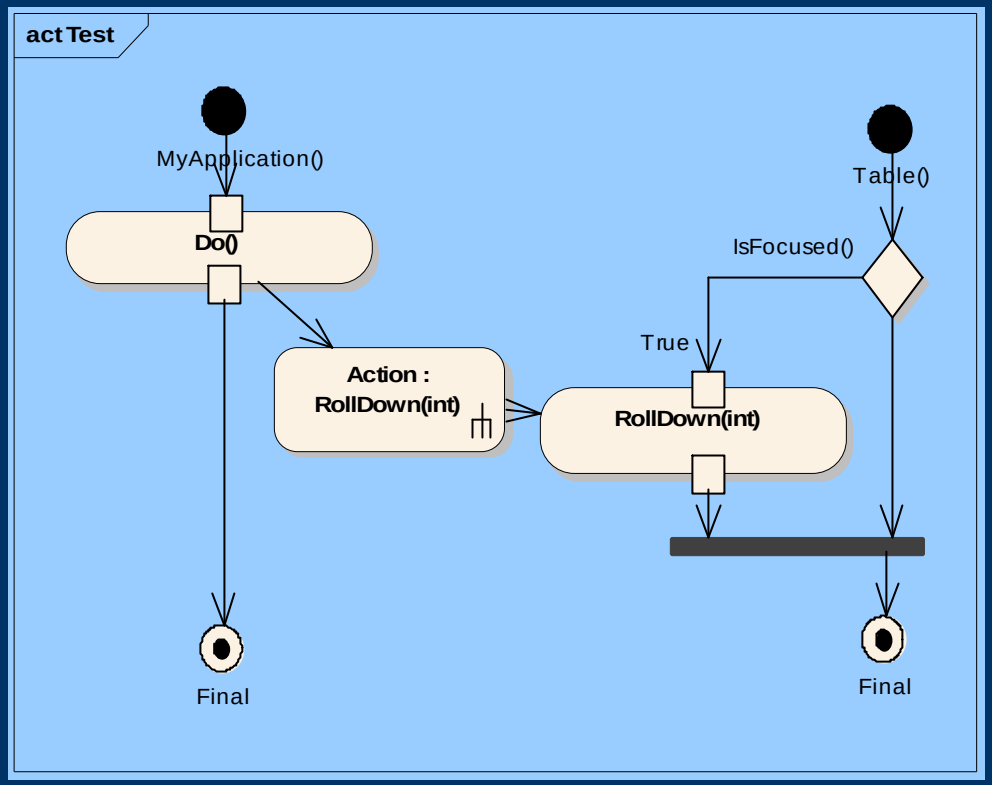
állapotdiagram (state machine diagram)

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



UML diagramok
aktivációs diagram (activity diagram)

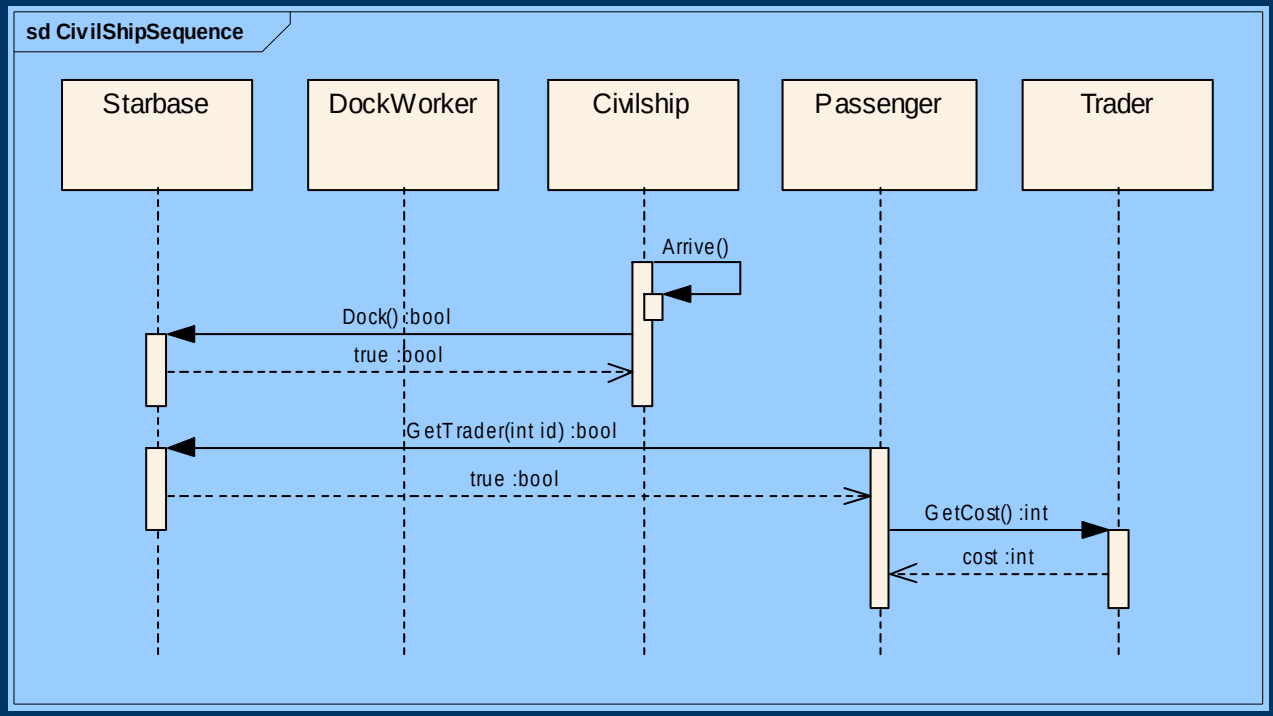
- A programfolyamatban keletkező aktivációkat kezeli, azaz, hogy milyen tevékenységeket hajt végre egy adott objektum a futás során



UML diagramok

szekvenciadiagram (sequence diagram)

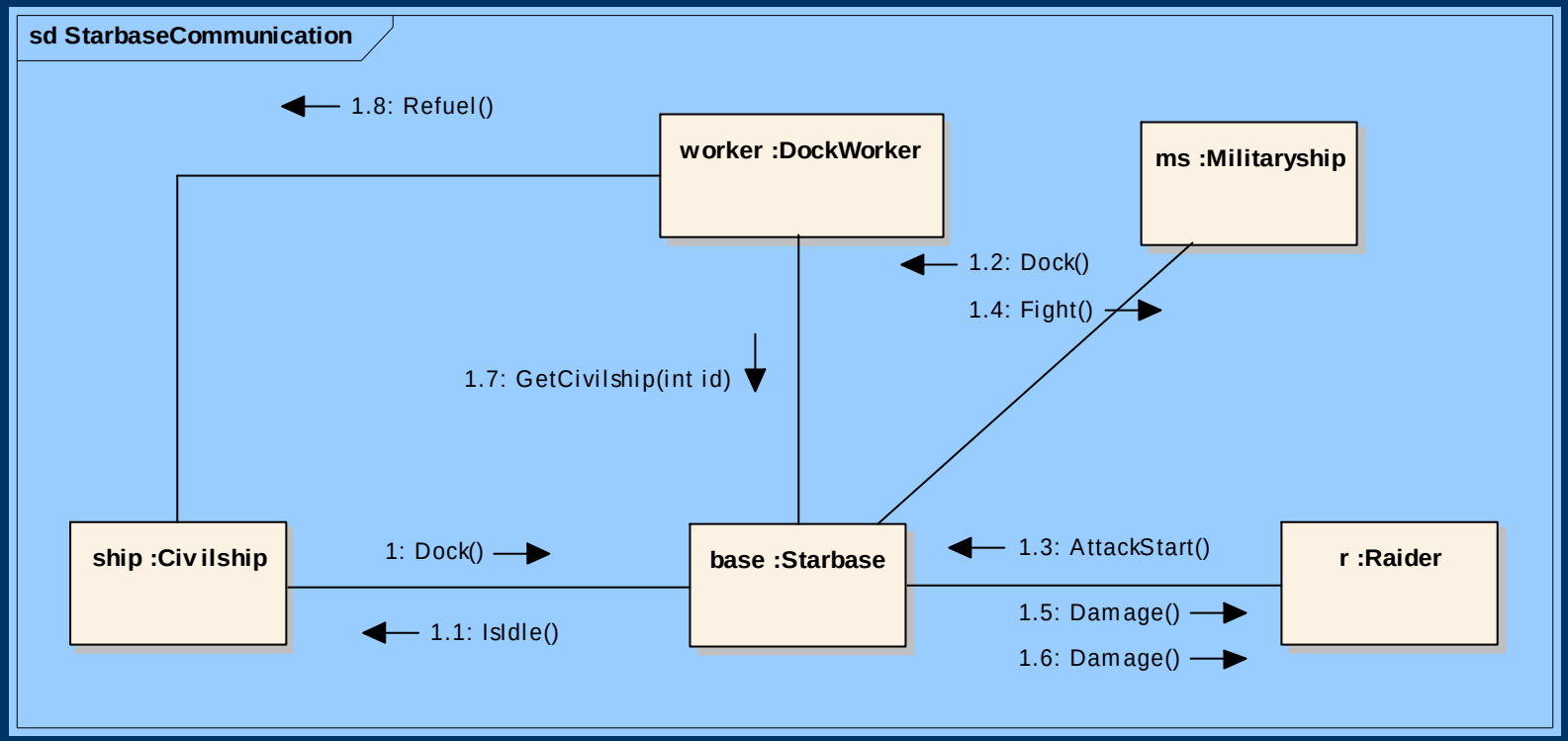
- Jelzi, hogy a program futása során az idő haladásával az egyes objektumok között milyen kommunikáció zajlik. Az üzenetátadások is megjelennek
- Megadja, mely objektumoknál van a vezérlés az adott időpillanatban



UML diagramok

kommunikációs diagram (communication diagram)

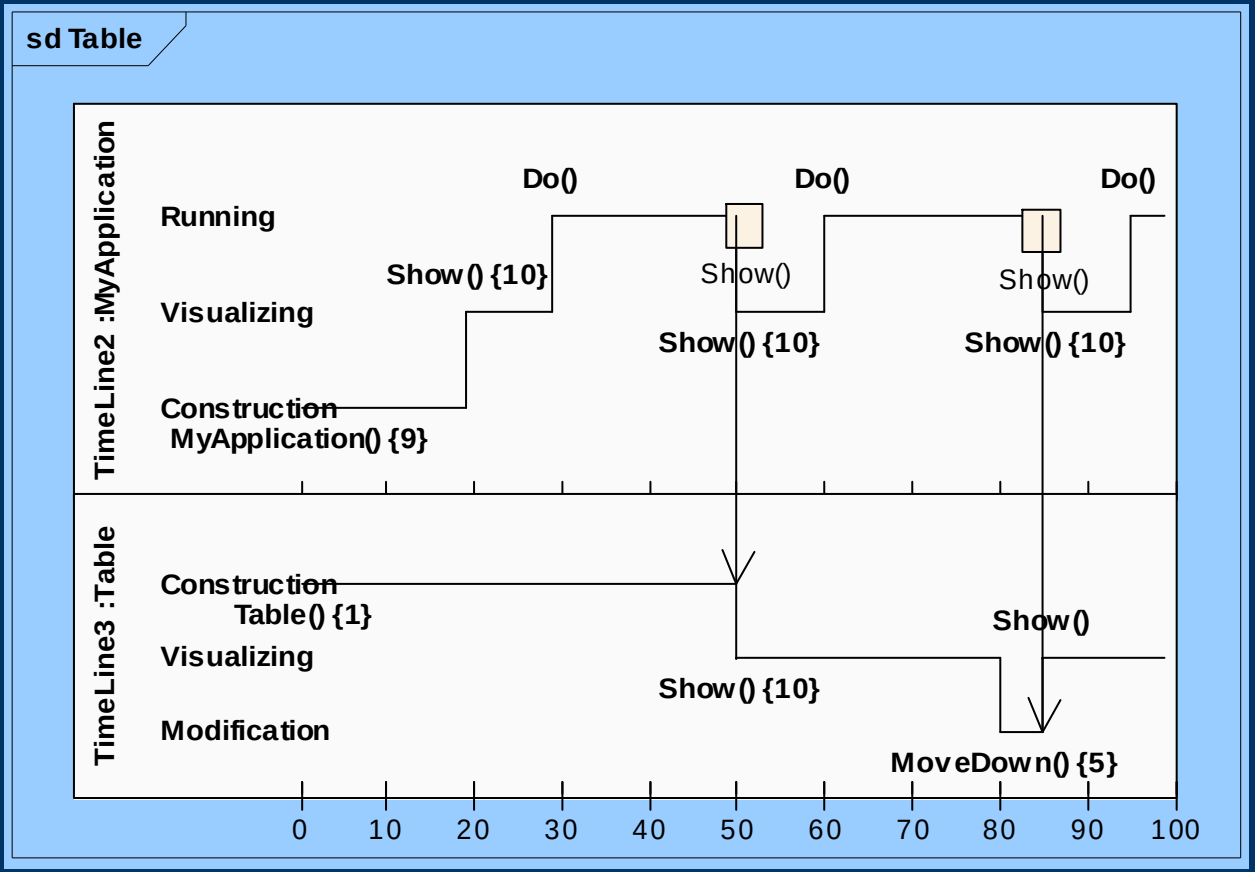
- Az objektumok közötti üzenetátadást és azok sorrendjét ábrázolja
- Az üzenetátadáson kívül megegyezik az objektum diagrammal



UML diagramok

időzítésdiagram (timing diagram)

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



UML diagramok

interakciós áttekintő diagram (interaction overview diagram)

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

A statikus modell
(szerkezeti modell)

- A modellezett szoftverrendszert a következő szempontok szerint jellemzi:
  - Milyen egységekből épül fel a rendszer
  - Mi ezeknek az egységeknek a feladata
  - Milyen kapcsolatban vannak egymással
- A rendszer objektumelvű szerkezetét írja le az osztálydiagram segítségével
- Az objektumdiagramok az osztálydiagramok példányait mutatják be
- Statikus modell = (osztálydiagram + osztályleírások) + (objektumdiagram + objektumleírások)

Objektumelvűség

- Az objektumelvű szoftverek működés közben objektumokat tartalmaznak
- Egy objektum a modellezett valóság egy alkotóelemének feleltethető meg
- Az objektumok saját feladatot látnak el, tartalmazhatnak és/vagy használhatnak más objektumokat
- Objektumelvű tervezés esetén a valóság fontos részeit úgy modellezzük, hogy a fenti alkotóelemeket meghatározva azokat objektumokként kezeljük, majd megadjuk a közöttük fennálló kapcsolatokat

Objektumelvűség

Példa szövegesen leírt objektumelvű modellre:

A buszközlekedés **állomások** között zajlik. Minden állomást jellemez a neve. Az állomások között **buszjáratok** közlekednek. Minden járatot egy szám azonosít és jellemzője a maximálisan szállítható utasok száma. Egy járat legalább két állomáson megáll, az állomások sorrendje rögzített. A buszközlekedésben **utasok** vesznek részt. Minden utasnak van neve, és egy kiinduló állomásról szeretne egy másik állomásra eljutni egy járattal (átszállás nélkül). Az utasok bizonyos időközönként megérkeznek a kiinduló állomásra, és ott várnak. Ha olyan buszra lehet felszállni az állomáson, amelyen még van hely, és meg fog állni az utas célállomásán, akkor az utas felszáll. A felszállás során előbb azok az utasok szállnak fel, akik előbb érkeztek az állomásra. Ha a busz megérkezik az utas célállomására, akkor az utas leszáll és befejezi tevékenységét. A buszjáratok menetrend szerint indulnak, azaz érkeznek meg az állomásokra. Egy állomásra érve előbb leszállnak a megfelelő utasok a buszról, majd felszállnak az utasok buszra. Felszállás után a járat a következő állomásra megy, ahová a menetrendben megadott időpontban érkezik. Végállomás esetén leszállás után a járat befejezi tevékenységét.

Az objektum informális definíciója I/II

- Az objektum azonosítható, az objektumok egymástól megkülönböztethetőek, függetlenül azok állapotától
  `String a = "abc", b = "abc";`
- Tulajdonságok/jellemzők/attribútumok tartoznak hozzá, köztük formális paraméter is lehet
  `student1.name = "John Smith"; student1.age = 23;`
- Állapottal rendelkeznek. Az attribútumok konkrét értékei együtt az objektum mindenkori állapotát határozzák meg
- Műveletek/események/függvények tartoznak hozzá
  `student1.goToSchool(elte);`

Az objektum informális definíciója II/II

- Korlátolt láthatósággal rendelkezik, azaz van látható és bizonyos körökben láthatatlan része (`public`, `private`, `protected`, csomag szintű láthatóság)
- Megkülönbözetünk absztrakt és konkrét megjelenési formát.Az absztrakt forma független a reprezentációtól és a megvalósítástól
- Az objektum az osztályának egy példánya. Objektum és osztálya közt az ú.n. „is a” reláció áll fenn
  `student1 instanceof Student`

Az osztály informális definíciója I/II

- Szerkezeti és viselkedésbeli jellemzőket tekintve hasonló tulajdonságú objektumok halmaza/típusa
- Az osztálynak van neve, melyet a hozzá tartozó összes objektum örököl
- Az osztálynak lehetnek attribútumai és paraméterei, melyek az objektumoknak is közös elemei
- Szolgáltatások/operációk/műveletek/függvények tartoznak hozzá
- Lehetnek osztályszintű attribútumok és műveletek, melyek az objektumoktól függetlenek

Az osztály informális definíciója II/II

- Az osztály lehet absztrakt. Egy absztrakt osztály megvalósítása hiányos vagy teljesen hiányzik, azaz létezik olyan operáció, melyhez megvalósítás nem tartozik, csupán absztrakt formában fogalmaztatik meg. Absztrakt osztályhoz nem tartozhat objektum
- Hasonlóan az objektumokhoz, az osztály láthatósága is szabályozható
- Az osztály lehet paraméteres (sablon- vagy generikus osztály). Közös működéssel bíró osztályok egy osztályát definiálja. A sablon alkalmazásakor a paraméter megadásával elkészül egy példány osztály és annak implementációja

Osztályok és objektumok JAVA-ban

Osztály definiálása

```
class Rectangle{  //név
    private int x1, y1, x2, y2;  //attribútumok
    public Rectangle(int x1, int y1,
        int x2, int y2){ //konstruktor
        this.x1 = x1; this.y1 = y1;
        this.x2 = x2; this.y2 = y2;
    }
    public int perimeter(){ //művelet
        return 2*(Math.abs(x1 - x2) +
            Math.abs(y1 - y2));
    }
}
```

Osztályok és objektumok JAVA-ban

- Objektumok példányosítása osztályból

```
Rectangle r1; //deklaráció
r1 = new Rectangle(0,0,4,5); //értékadás
final Rectangle r2 = new Rectangle(3,4,
    10,-1); //definíció/inicializálás
```

- Objektumok műveleteinek használata

```
int p = r1.perimeter();
```

- Az objektumok memóriabeli tárolása referencia szerint történik
- Nyílt rekurzió: **`this`** kulcsszó

Köszönöm a figyelmet!

Kérdések?