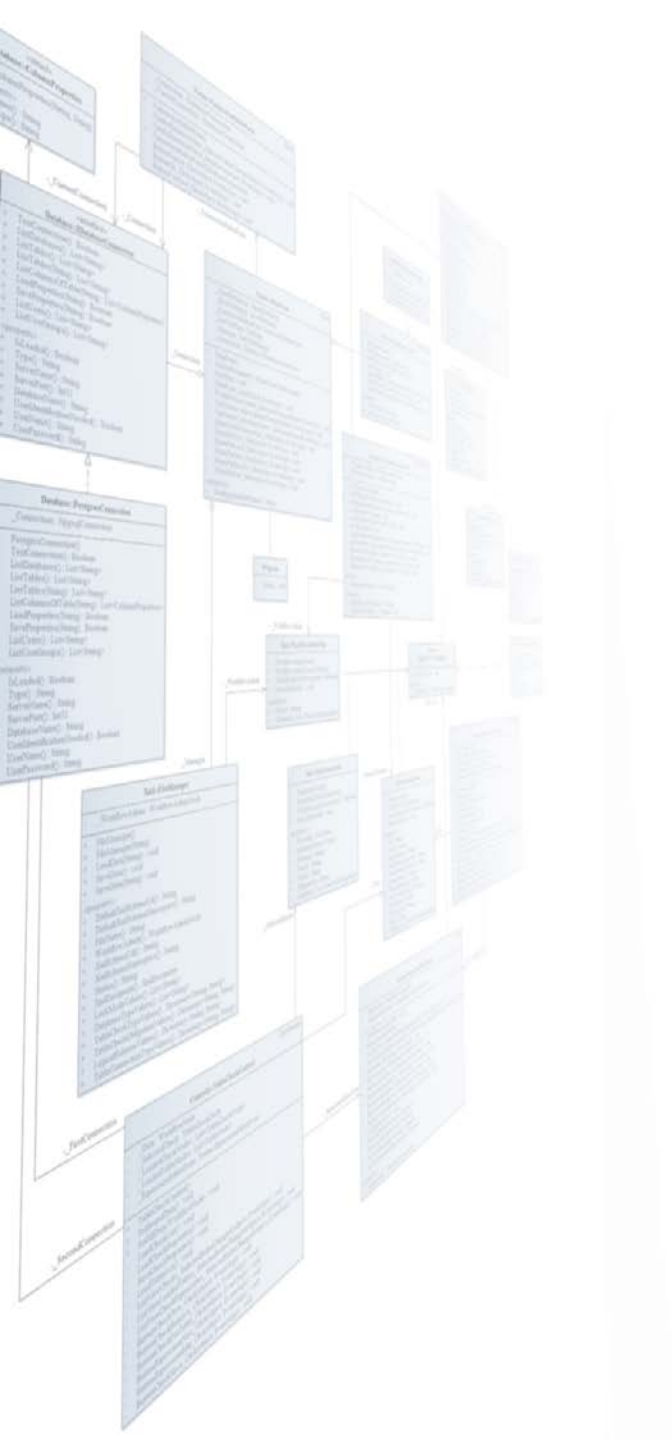


Pázmány Péter Katolikus Egyetem
Információs Technológiai és Bionikai Kar

Bevezetés a Programozásba II

5. előadás

Objektumorientált programozás és tervezés

© 2014.03.10. Giachetta Roberto
groberto@inf.elte.hu
http://people.inf.elte.hu/groberto

Objektumorientált programozás

Kialakulása

- A procedurális programozási paradigma összetett alkalmazások esetén számos korlátozást tartalmaz:
  - a program nem tagolható kellő mértékben
  - az adatok élettartama nem eléggé testre szabható
  - a feladat módosítása utóhatásokkal rendelkezhet
- Ezek megoldása a *felelősség továbbadása*
  - *programegység*eket alakítunk ki, amely rendelkeznek saját adataikkal és műveleteikkel, ezeket egységbe zárjuk, megvalósításukat elrejtjük
  - a feladat megoldását a programegységek együttműködésével, *kommunikáció*jával valósítjuk meg

Objektumorientált programozás

Objektumok

- *Objektum*nak (*object*) nevezzük a feladat egy adott tárgyköréért felelős programegységet, amely tartalmazza a tárgykör megvalósításához szükséges adatokat, valamint műveleteket
  - az objektum működése során saját adatait manipulálja, műveleteit futtatja és kommunikál a többi objektummal
  - pl.: egy téglalapot kezelhetünk objektumként
    - adatai: szélessége és magassága
    - műveletei: területlekérdezés, kirajzolás
  - pl.: egy egyetemi hallgatót kezelhetünk objektumként
    - adatai: neve, azonosítója, kurzusai
    - műveletei: kurzus felvétele, teljesítése

Objektumorientált programozás

Objektumok állapotai

- Az objektumok *állapot*tal (state) rendelkeznek, ahol az állapot mezőértékeinek összessége
  - két objektum állapota ugyanaz, ha értékeik megegyeznek (ettől függetlenül az objektumok különbözőek)
  - az állapot valamilyen *esemény* (műveletvégzés, kommunikáció) hatására változhat meg
- A program teljes állapotát a benne lévő objektumok összállapota adja meg
- Az objektumok *életciklus*sal rendelkeznek, létrejönnek (a konstruktorral), működést hajtanak végre (további műveletekkel), majd megsemmisülnek

Objektumorientált programozás

Objektum-orientált programok

- *Objektum-orientált*nak nevezzük azt a programot, amely egymással kommunikáló objektumok összessége alkot
  - minden adat egy objektumhoz tartozik, és minden algoritmus egy objektumhoz rendelt tevékenység, nincsenek globális adatok, vagy globális algoritmusok
  - a program így kellő tagoltságot kap az objektumok mentén
  - az adatok élettartama így összekapcsolható az objektum élettartamával
  - a módosítások általában az objektum belsejében véghezvihetők, ami nem befolyásolja a többi objektumot, így nem szükséges jelentősen átalakítani a programot

Objektumorientált programozás

Objektum-orientált programok

- Az objektum-orientáltság öt alaptényezője:
  - *absztrakció*: az objektum reprezentációs szintjének megválasztása
  - *enkapszuláció*: az adatok és alprogramok egységbe zárása, a belső működés elrejtése
  - *nyílt rekurzió*: az objektum mindig látja saját magát, eléri műveleteit és adatait
  - *öröklődés*: az objektum tulajdonságainak átruházása más objektumokra
  - *polimorfizmus és dinamikus kötés*: a műveletek futási időben történő működéshez kötése

Objektumorientált programozás

Osztályok

- Az objektumok viselkedési mintáját az *osztály* tartalmazza, az osztályból *példányosít*hatjuk az objektumokat
  - tehát az osztály az objektum típusa
- Az osztályban tárolt adatokat *attribútum*oknak, vagy *mező*knek (*field*), az általa elvégezhető műveleteket *metódus*oknak (*method*) nevezzük, együtt ezek az osztály *tag*jainak (*member*)
- Az osztály tagjainak szabályozhatjuk a láthatóságát, a kívülről látható részét *interfész*nek, a kívülről nem látható részét *implementáció*nak nevezzük
  - a metódusok megvalósítása az implementáció része, tehát más osztályok számára a működés mindig ismeretlen

Objektumorientált programozás

Osztályok megvalósítása

- Az osztályokat minden nyelv más formában valósítja meg, de az általános jellemzőket megtartja
- A C++ programozási nyelv támogatja az objektumorientált programozást, noha alapvetően procedurális
- A C++ osztály szerkezete:

```
class/struct <osztálynév> {
public/private:
    <típus> <mezőnév>;
    …
    <típus> <metódusnév> ([ <paraméterek> ]) { … }
    …
};
```

Objektumorientált tervezés

Szoftverek tervezése

- A program szerkezetének és működésének megtervezések az osztályok és objektumok szempontjából történik
  - a szerkezeti tervezésnél az osztályok tagjait, azok kapcsolatait, illetve elhelyezkedését a programban, ez alkotja a *statikus terv*et
  - a programfutási tervezésnél az osztályok időbeli működését, az objektumok állapotainak változását modellezzük, ez a *dinamikus terv*ezés
- Az objektumorientált tervezés eszköze a *Unified Modeling Language* (UML), amelyben 13 diagramtípus segítségével tervezhető meg a program szerkezete és működése

Objektumorientált tervezés

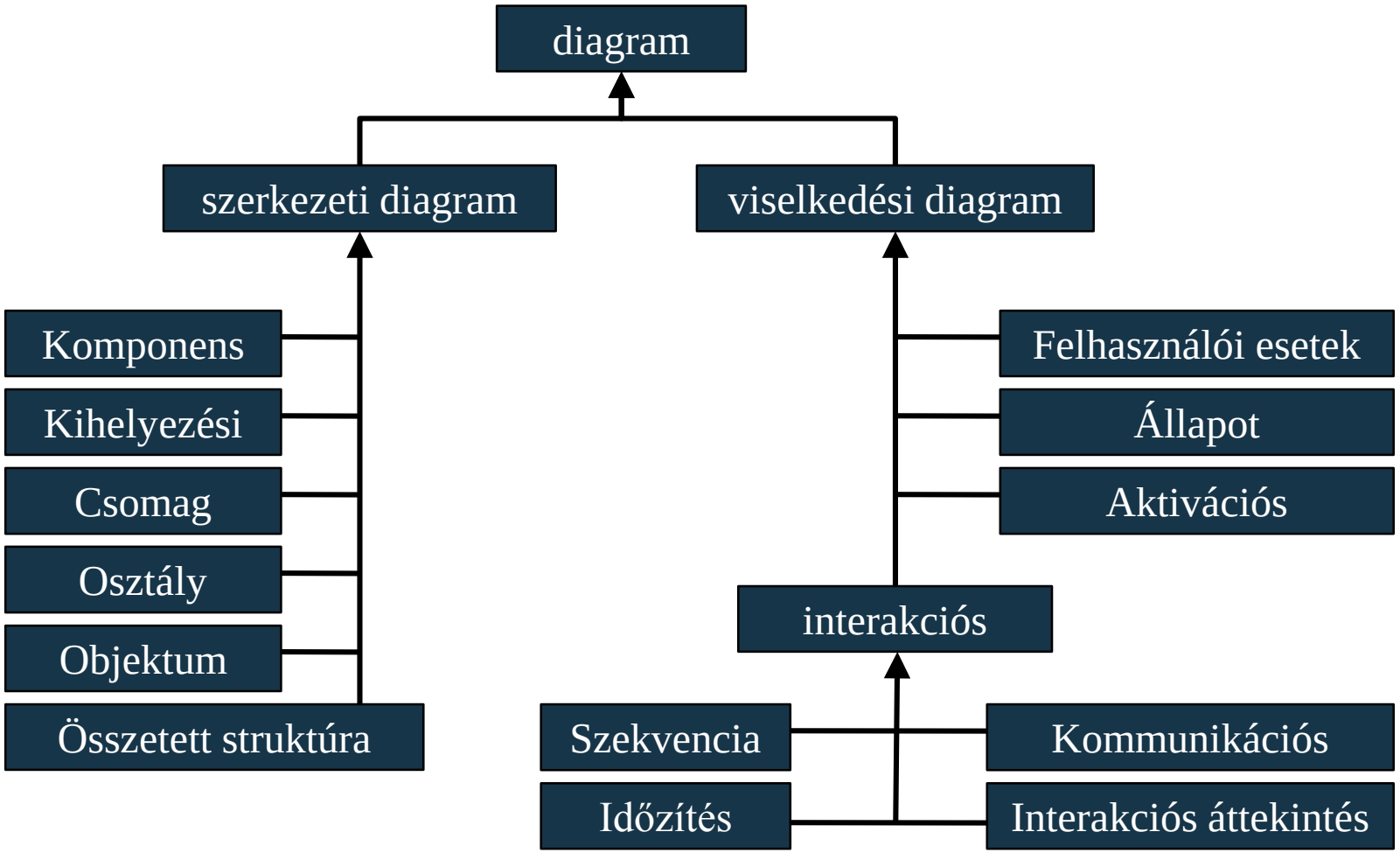
Az UML nyelv

- 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
- A legfrissebb változat a 2.4.1-es szabvány

- 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

Objektumorientált tervezés

Az UML nyelv



Objektumorientált tervezés

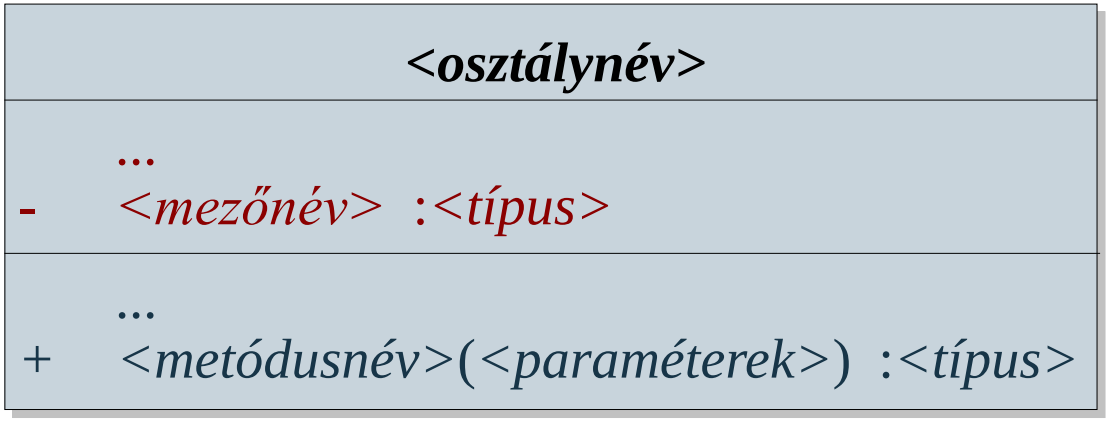
Az UML nyelv

- A szoftverfejlesztés különböző szakaszaiban az UML különböző diagramjait kell alkalmaznunk:
  1. elemzés: felhasználói esetek, komponens, kihelyezési
  2. tervezés:
     - statikus tervezés: csomag, osztály, objektum, komponens
     - dinamikus tervezés: állapot, szekvencia, aktivációs, interakciós áttekintési, kommunikációs
  3. tesztelés: időzítés
- A későbbi fázisokban a korábban létrehozott diagramok újra alkalmazhatóak, tovább finomíthatóak

Objektumorientált tervezés

Az osztálydiagram

- Az *osztálydiagram* a programban szereplő osztályok szerkezetét, és a közöttük lévő kapcsolatokat definiálja
  - az osztálynak megadjuk a nevét, valamint mezőinek és metódusainak halmazát (típusokkal, paraméterekkel)
  - megadjuk a tagok láthatóságát látható (**+**), illetve rejtett (**-**) jelölésekkel



Objektumorientált tervezés

Példa

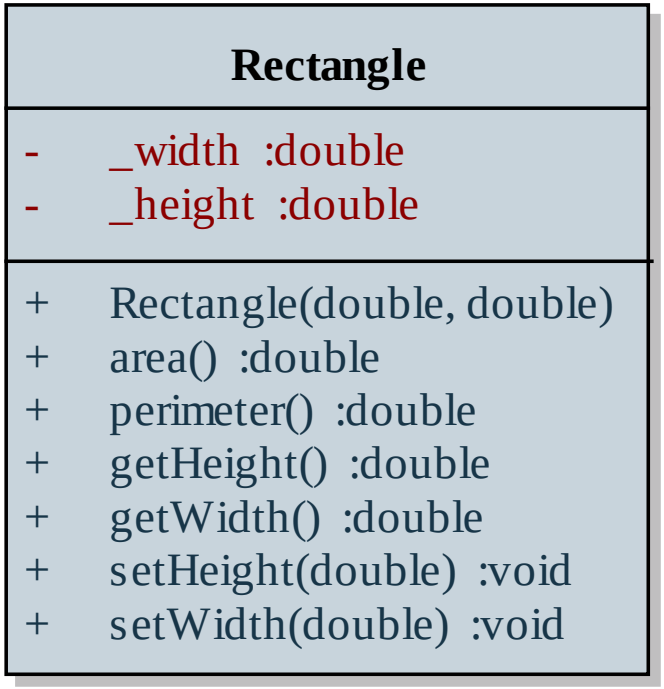
Feladat: Valósítsuk meg a téglalap (**`Rectangle`**) osztályt, amely utólag átméretezhető, és le lehet kérdezni a területét és kerületét.

- a téglalapnak a feladat alapján elég a méreteit letárolni (**`_height`**, **`_width`**), amelyek valós számok lesznek
- ezeket a későbbiekben lekérdezhetjük (**`getWidth()`**, **`getHeight()`**), vagy felülírhatjuk (**`setWidth(double)`**, **`setHeight(double)`**)
- lehetőséget adunk a terület, illetve került lekérdezésére (**`area()`**, **`perimeter()`**)
- lehetőséget adunk a téglalap létrehozására a méretek alapján (**`Rectangle(double, double)`**)

Objektumorientált tervezés

Példa

Tervezés:



Objektumorientált tervezés

Példa

Megoldás:

```
struct Rectangle // téglalap típusa
{
private:
    double _width; // szélesség
    double _height; // magasság

public: // látható rész
    Rectangle(double w, double h) { … }
       // 2 paraméteres konstruktor művelet
    double area(); // téglalap területe
    double perimeter(); // téglalap kerülete
    …
};
```

Objektumorientált tervezés

A tervezési folyamat

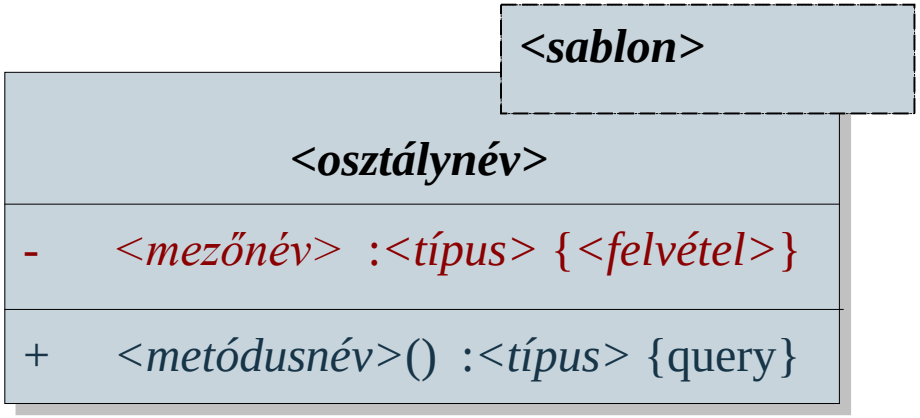
- A tervezés során a program szerkezetét általában iterálva, több lépésben határozzuk meg
  1. elemezzük a feladatot, feltérképezzük az igényeket, követelményeket, a felhasználói eseteket
  2. a követelményekben azonosítjuk a lehetséges tárgyköröket, amelyekhez az osztályokat rendeljük
  3. megtervezzük az osztályok felületét, kapcsolódási pontjait más osztályokhoz
  4. megtervezzük az osztályok belső reprezentációját, működési elveit (algoritmusait)
  5. megtervezzük az osztályok viselkedési módját, végrehajtásai folyamatát, állapotváltozásait

} iterálva

Objektumorientált tervezés

Az osztálydiagram

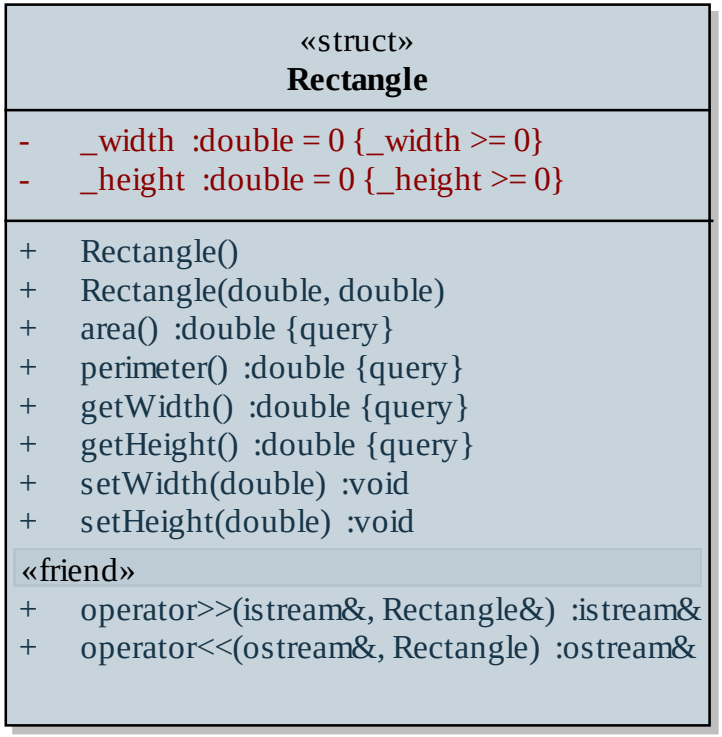
- A lépések során az osztálydiagramot is tovább pontosítjuk
  - feltételeket szabhatunk a mezőkre, metódusokra a **{…}** jelzéssel (speciálisan a lekérdező műveleteket a **{query}** jelzéssel jelölhetjük)
  - további tulajdonságokat jelölhetjük, illetve csoportba foglalásokat végezhetünk a **<<…>>** jelzéssel



Objektumorientált tervezés

Példa

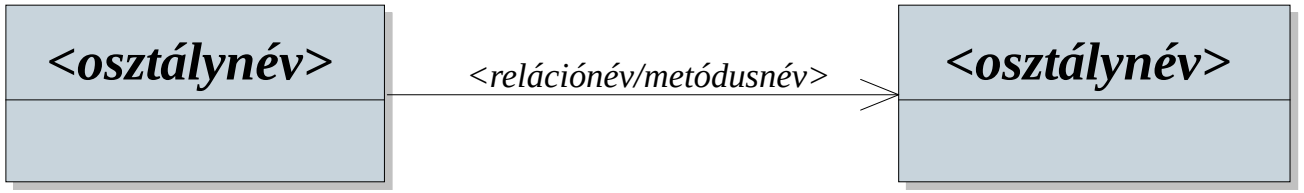
Feladat: Valósítsuk meg a téglalap (**`Rectangle`**) osztályt, amely utólag átméretezhető, és le lehet kérdezni a területét.



Objektumorientált tervezés

Objektumok közötti kapcsolatok

- Az alkalmazásokban rendszerint több osztály szerepel, amelyek objektumai kommunikálhatnak egymással, ezért az osztályok között kapcsolatokat állíthatunk fel
  - a legegyszerűbb kapcsolat az *egyszerű kommunikáció* (*asszociáció*)
    - az osztály meghívja más osztály (látható) metódusát, vagy bármilyen módon hivatkozik rá a műveletek végrehajtása során
    - a kapcsolat a célosztály felé irányított



Objektumorientált tervezés

Példa

Feladat: Készítsünk grafikus felületű alkalmazást, amelyben a képernyő tetszőleges pontján el tudunk helyezni egy új téglalapot (bal kattintással), majd annak színét tudjuk módosítani (jobb kattintással), amelyiket elhelyezkedik az egér.