OOP

Elmélet: Angster E. – Objektumorientált tervezés és programozás

Gyakorlat: Levendovszky T., Benedek Z. – Szoftverfejlesztés C++ nyelven

K. Sierra, B. Bates – Agyhullám: Java

Objektumorientált paradigma

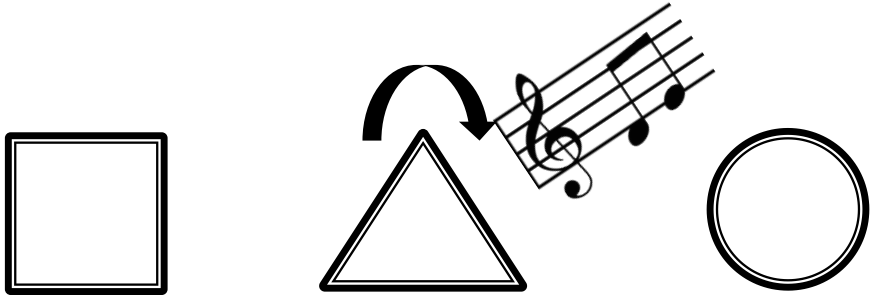
Hibrid nyelv(C++) lehetővé teszi, tisztán OO nyelv (C#, Java) kikényszeríti az alábbiak használatát:

- Egységbezárás / adatrejtés – Encapsulation / Data hiding
- Öröklődés (származtatás) - Inheritance
- Polimorfizmus – Polymorphism

OOP: a program legkisebb eleme az objektum, melyek egymás között kommunikálnak (üzenet küldése és fogadása)

Procedural style vs. OOP

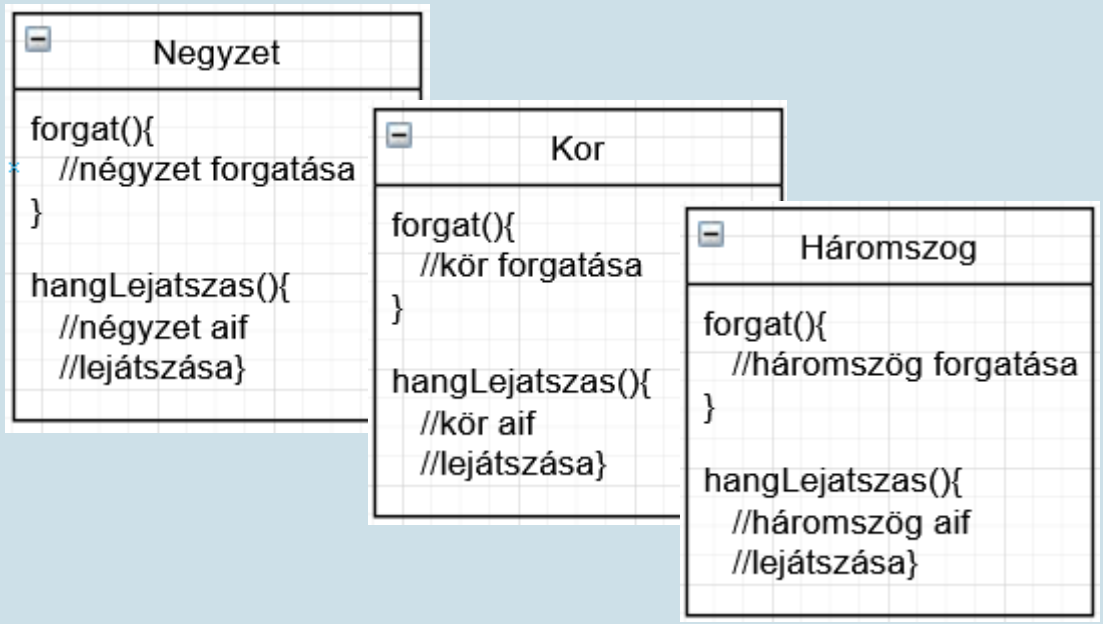
- Terv: grafikus felületen alakzatok, amire kattintva elfordulnak 360 fokkal és lejátszák a megfelelő .aif hangfájlt.



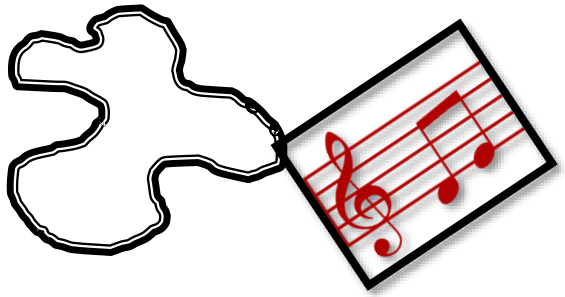
- Metódusokkal: mit kell csinálni?
- Objektumokkal: milyen dolgok vannak a programban?

Készül a kód…

Metódusokkal	Osztályokkal

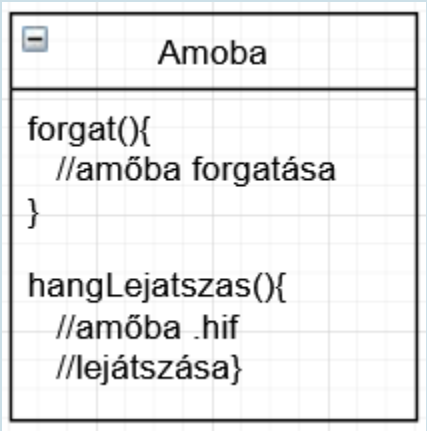
forgat(alakzatSzama){
//forgatás 360 fokkal
}
 	
hangLejatszas(alakzatSzama){
//alakzatSzama adja meg,
//melyik hangfájlt kell
//lejátszani
	

Változik a terv…

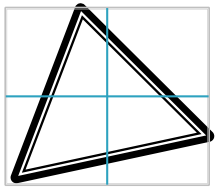


- Lesz egy amőba is, azt tudja, amit a többi alakzat, de .hif hangfájlt játszik le

Metódusokkal	Osztályokkal

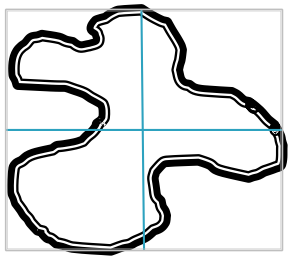
hangLejatszas(alakzatSzama){
//ha nem amőba
//alakzatSzama adja meg,
//melyik hangfájlt kell
//lejátszani
//else
//amőba .hif lejátszás
	

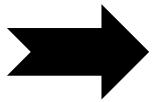
Változik a meglévő kód!!!	A meglévő nem változik, csak lesz egy új osztály

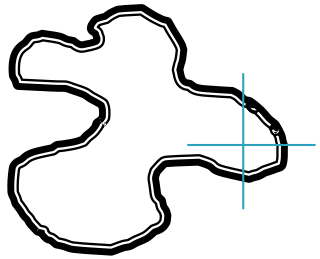


Kiderül, hogy…

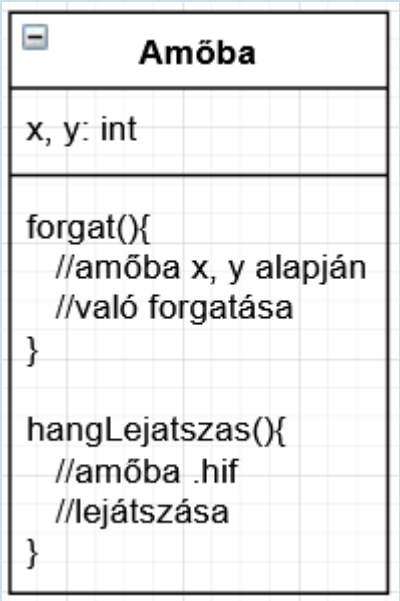
- Az amőba máshogy forog:





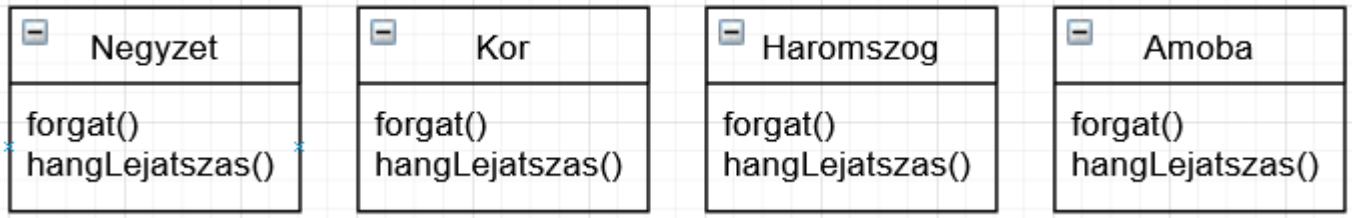


Metódusokkal	Osztályokkal

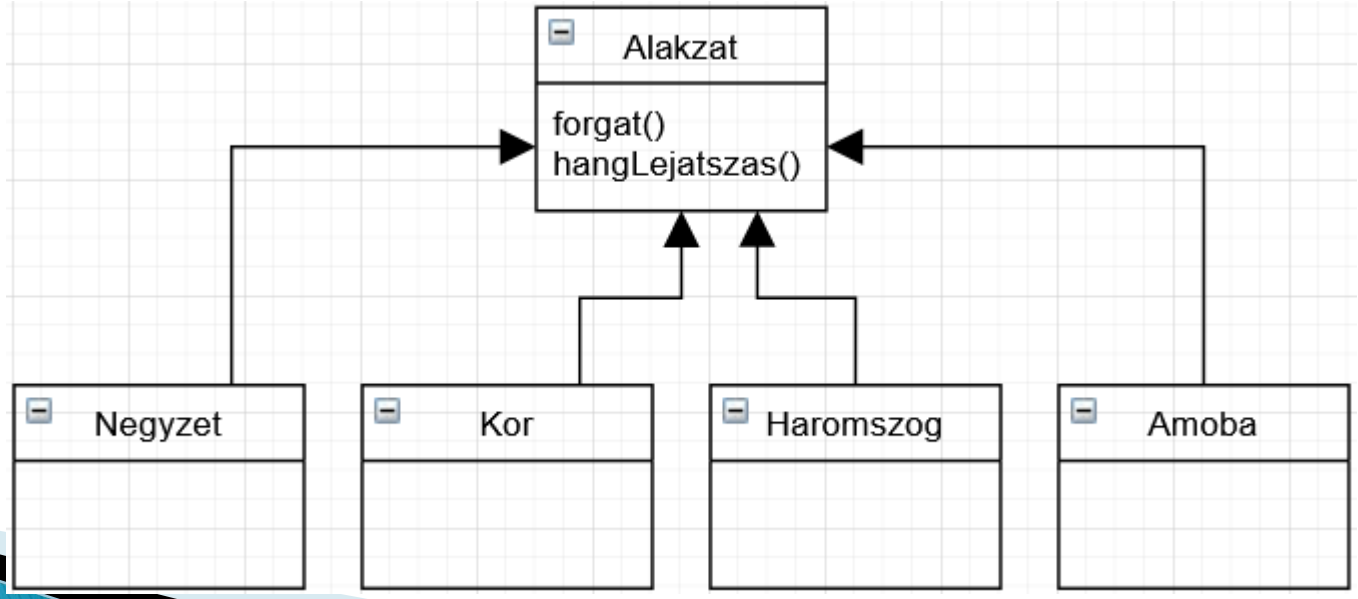
forgat(alakzatSzama, x, y){
//ha nem amőba
//négyzet alapján
//középpont , aztán
//forgatás
//else
//x és p eltolással
//forgatunk
}
	

Ami korábban működött, annak annyi!!!	Ami működött, ahhoz nem kell nyúlni, csak az amőbát kell átírni

Ez kódduplikálás!



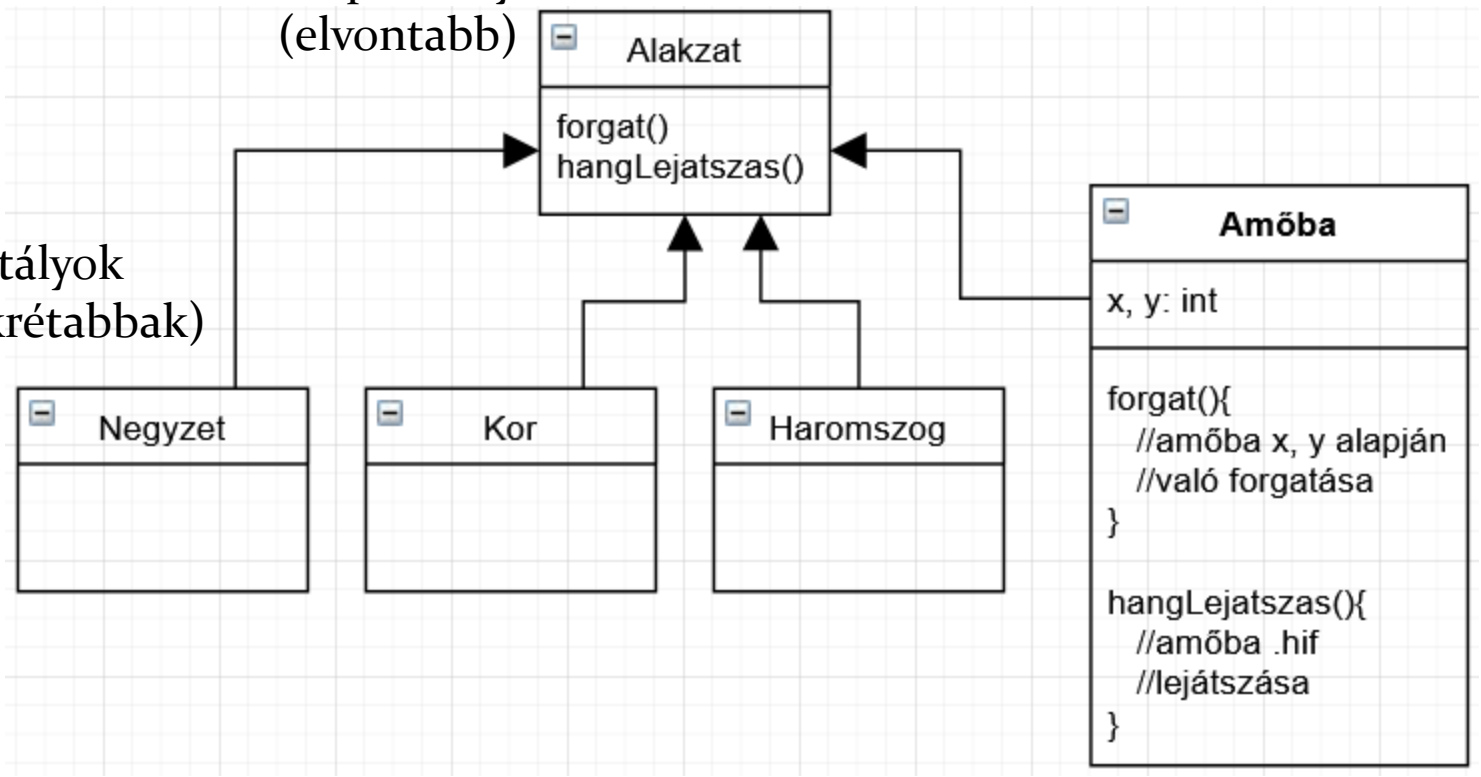
- Mi a közös a 4 osztályban?
  - mind alakzat, mind forog, mind hangot játszik le
- A közös alapból öröklődjenek a konkrét alakzatok!



Az amőba másmilyen!

Alaposztály
(elvontabb)

Alosztályok
(konkrétabbak)



Az Amoba osztályban felülírtuk (override) az Alakzat osztály tagfüggvényeit!

Osztály és objektuma(i)/példánya(i)

- Egy osztály: az objektumok tervrajza
- Sok objektum: mind saját értékekkel rendelkezik


```
class Kutya{
    int méret;
    String fajta, név;
    void ugat(){
        System.out.println("vau");
    }
}
```

Egy osztály (class)

```
public static void main(String[] args) {
    Kutya k = new Kutya();
    k.méret = 40;
    k.ugat();
}
```

```
class Film{
    int pont;
    String cim, kategoria;
    void lejatszas (){
        System.out.println("film lejátszása");
    }
}
```

```
public static void main(String[] args) {
    Film elso = new Film();
    elso.cim = "Nem tudok programozni";
    elso.kategoria = "tragédia";
    elso.pont = -2;
    Film masodik = new Film();
    masodik.cim = "Procedural vs. OOP";
    masodik.kategoria = "vígjáték";
    masodik.pont = 10;
    masodik.lejatszas();
    Film harmadik = new Film();
    harmadik.cim = "UML földi halandóknak";
    harmadik.kategoria = "könyv";
    harmadik.pont = 3;
}
```

Szövegből UML: főnév, ige

- Feladatspecifikáció:

Gyuszinak van egy 7x5 m-es terepasztala, amelyen kisautók robognak különböző irányokban. Minden autóra jellemző annak színe és sebessége. Ha egy autó a terepasztal falához ér, ott rugalmasan ütközik, és a megváltozott irányban folytatja útját. Gyuszi úgy játszik, hogy kedve szerint egy-egy autót megfordít vagy arrébb tol. Előbb vagy utóbb Gyuszi befejezi a játékot – ha ráüt az asztalra, az autók megállnak.

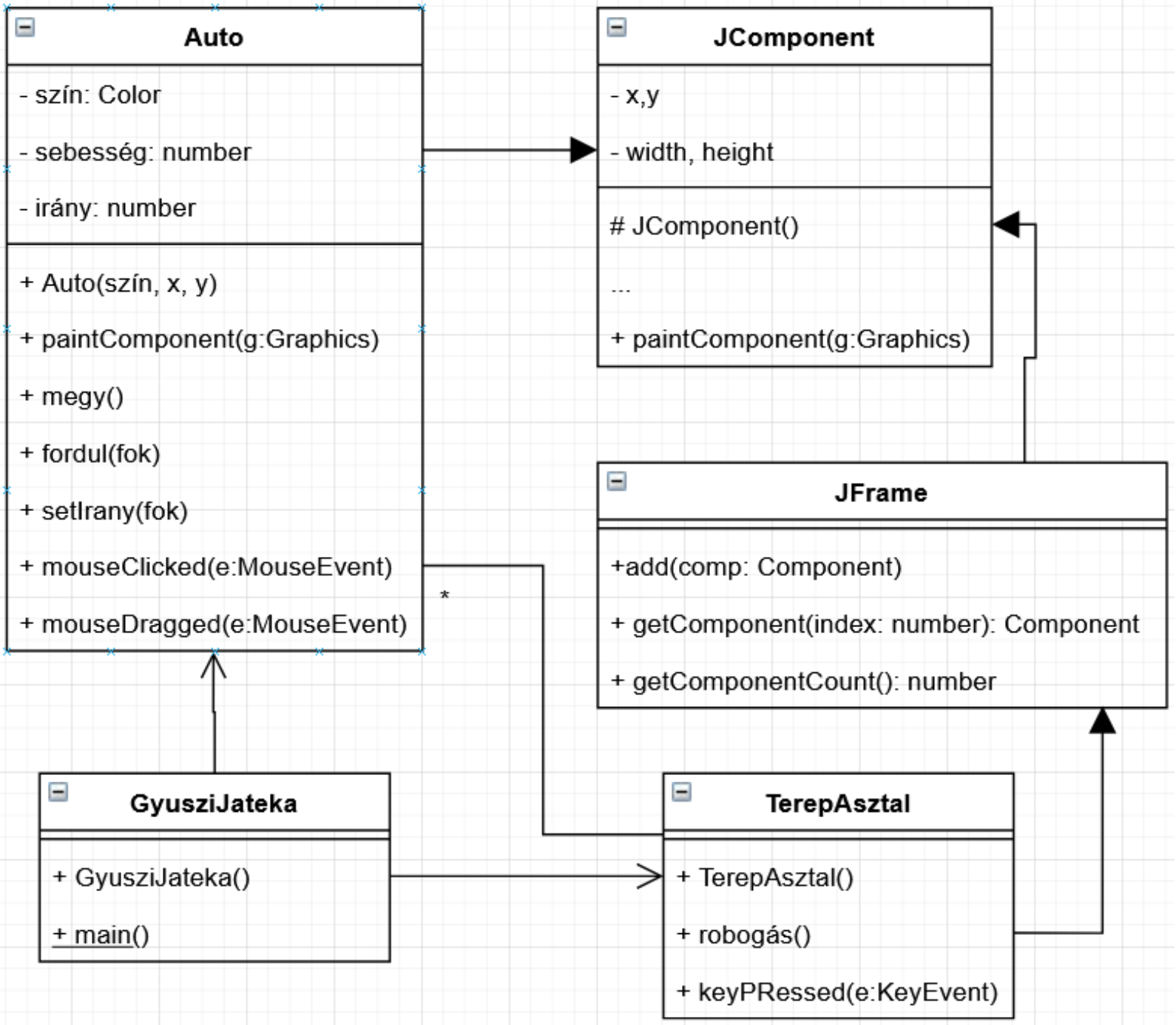
TFH.: az autók nem ütköznek, átmennek egymáson!

- Főnevek
  - **Gyuszi** = aktor, a prg használója
  - **Terepasztal** = asztal = GUI
  - Kisautó = **autó**
  - **Irány** (az autó pillanatnyi tulajdonsága)
  - **Szín** (az autó állandó tulajdonsága)
  - **Sebesség** (az autó állandó tulajdonsága)
  - Fal = terepasztal széle
  - Út = az autók **pozíció**jának változása
  - Játék = **Gyuszi játéka** = feladat = program
- Igék
  - Robog = **megy**
  - Ér = **ütközik**
  - Játszik (fut a program)
  - **Megfordít** – használati eset
  - **Arrább tol** – használati eset
  - Befejezi = ráüt = **vége a programnak**
  - **Megállnak** (a játék végén)
  - Átmennek = nem ütköznek – ezt nem kell kezelni

Objektumok, felelősségek

- Gyuszi: aktor. Kívülről fogja manipulálni a programot
- Terepasztal
  - Tulajdonság: méret
  - Felelősség: összetartja és működteti az autókat
- Autó
  - Tulajdonság: pozíció, szín, sebesség, irány
  - Felelősség: megy vmekkora sebességgel, megfordul, arrébb megy
- Gyuszi játéka
  - Tulajdonság: -
  - Felelősség: ez az objektum vezérli az egész programot

Osztály-diagram



Főnév, ige, UML

Osztály neve

Adattagok (korábban változók), az objektumok állapotát mutatják

Tagfüggvények (korábban metódusok), az objektumok „lehetősége”

Osztály - szereplő

- Adattagok – jellemző a szereplőre
- Konstruktorok – az adattagok milyen kezdő értékkel legyenek
- Getterek ? – mi az, amit meg akarunk tudni a szereplőről
- Setterek ??? – mi az, amit átállítunk
- Overrride – mi az, amt máshogy csinál, mint amit örökölt
- Saját tagfüggvények – mi az, ami saját „lehetőség”

Kitalálós játék

- A játékban három - gépi - játékos próbál kitalálni egy számot.

- JatekIndito:
  - main() //játék indítása
- Jatek:
  - referenciák a játékosokra
  - start()
- Jatekos:
  - tipp, tippel()

```
Gondoltam egy számra, 0 és 9 között...
a kitalálandó szám: 5
tippelésem: 2
tippelésem: 8
tippelésem: 8
1. játékos tippje: 2
2. játékos tippje: 8
3. játékos tippje: 8
a játékosok újra próbálkoznak
a kitalálandó szám: 5
tippelésem: 2
tippelésem: 6
tippelésem: 5
1. játékos tippje: 2
2. játékos tippje: 6
3. játékos tippje: 5
Van nyertes!
1. játékos tippje jó?false
2. játékos tippje jó?false
3. játékos tippje jó?true
Játék vége
```