III. OOP (objektumok, osztályok)

1. Természetes emberi gondolkozás

Az Objektumorientált paradigma alapelvei nagyon hasonlítanak az emberi gondolkozásra. Érdemes ezért elsőként az emberi gondolkozás elveit megismernünk.

Az embert a gondolkozás során a következő gondolatok vezérlik:

- **Absztrakció**
  A világot leegyszerűsítjük, mert annak minden részletét túlságosan bonyolult lenne feldolgozni.
- **Osztályozás**
  A dolgokat tulajdonságaik, viselkedésük alapján kategóriákba soroljuk.
- **Megkülönböztetés**
  A dolgok különböznek egymástól. Nincs két egyforma objektum.
- **Általánosítás/Specializáció**
  Az objektumok között keressük a hasonlóságot és a különbözőséget
- **Kapcsolatok**
  Az objektumokat megpróbáljuk kapcsolni egymáshoz pl.: sütemény alkotóelemei
  - ***Ismertségi kapcsolat***
    - az objektumok függetlenek egymástól (pl.:kutya-ház)
  - ***Egész-rész kapcsolat***
    - az objektumok függenek egymástól (pl.: kutya és a 4 lába)

2. OOP

Az Objektum-orientált programozás (Object Oriented Programming) nem új dolog. Már az 1960-as években foglalkoztak az elveivel. Elterjedése mégis csak a 70-es években történt a XEROX cégnek köszönhetően.

Definíció: *Egy OOP egymással kommunikáló objektumok összessége.*

3. Objektum

Az objektum információ tárolásával foglalkozik, és kérésre feladatot hajt végre. Leegyszerűsítve azt is mondhatnánk, hogy egy objektum adatok és metódusok összessége. Ha párhuzamot szeretnénk vonni a valós élet és az objektumorientáltság között, akkor azt mondhatnánk, hogy minden dolog, amit a

környezetünkben tapasztalunk (és maga a környezet is) objektum. A természetes gondolkozás elveinek megfelelően minden objektum egyedi. Azonban az azonos tulajdonságokkal rendelkező objektumokat osztályokba soroljuk. A Java nyelvben is először osztályokat fogunk definiálni, és az osztályok példányosításával fogjuk létrehozni az egyedi objektumokat.

4. Osztályok

Tehát az objektumokat osztályokba soroljuk. Valójában az osztályokban fogjuk definiálni a tulajdonságokat (adatokat) és a viselkedést (metódusokat), amik a példányosítás révén fognak az objektumok egyedi jellemzőivé válni.

Példa az osztályozásra:

- Állat
  - Háziállat
    - Macska
    - Kutya
  - Vadállat
    - Oroszlán

A fenti ábráról eszünkbe juthat, hogy a szemantikus hálóknál is hasonló dolgokról tanultunk.

Példányosításnak hívjuk azt a folyamatot, amelynek során egy osztályból létrehozunk egy objektumot, amit a későbbiekben már példánynak fogunk hívni. Dolgozni csak a példányokkal fogunk tudni.
A könnyebb érthetőség kedvéért nézzünk meg egy példát a valós életből.

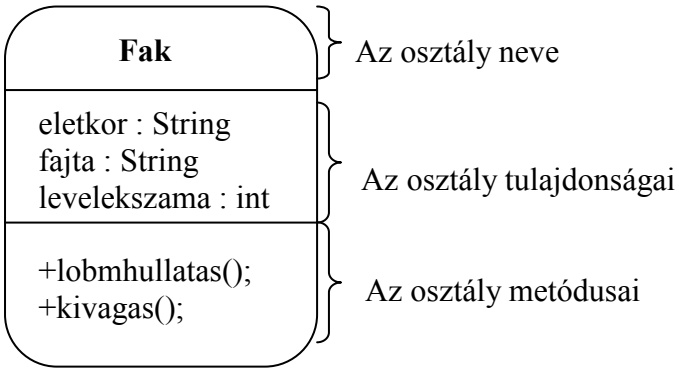
Példa:
Tekintsük a fákat. Mindenki tudja, hogy a fáknak milyen tulajdonságaik vannak: van levelük, törzsük, tudjuk az életkorukat …stb. Tudjuk, hogy a fák nőnek, kivágják őket…stb.
A fák egy osztály, ami két dologból áll:

- tulajsondágokból:
  - fajta
  - életkor
  - magasság …stb
- viselkedésből (metódusokból)
  - növekedés()
  - lombhullatás()
  - kivágás() …stb.

Ezek a tulajdonságok és metódusok nagyjából minden fára igazak. Azonban dolgozni mindig csak egy-egy konkrét fával tudunk. Tehát az osztályok csak a fák elméleti leírását végzik, de amikor pl.: a tölgyfát vesszük figyelembe a kertünkben, akkor az fizikai valóságában létezik, konkrét magassága van, ősszel tapasztaljuk, hogy hullatja a leveleit …stb. Ugye mindenki érzi, hogy utóbbi esetben nem osztályról, hanem egy konkrét objektumról, az osztály egy példányáról van szó.

Így működik ez a Java nyelvben is.
Amikor programot írunk, akkor osztályokat hozunk létre és azokat használjuk fel. Mint azt az elmúlt órán is láttuk egy osztályt biztosan létre kell hoznunk, amelyből lehetőségünk lesz más osztályokat is használni.

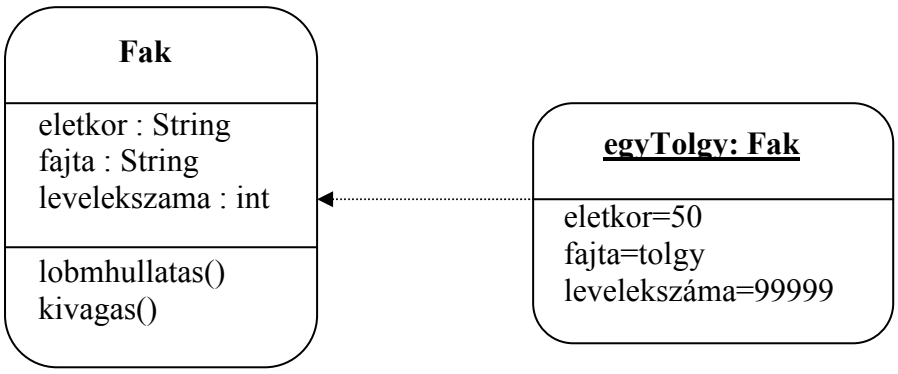
Az **UML** (**U**nified **M**odelling **L**anguage) az objektumorientált modellezés alapnyelvévé vált.
Az UML jelölésrendszerben a Fák osztály a következőképpen néz ki:



A nyelvben az elnevezési konvenciók a következők:

- Az osztály nevét nagy kezdőbetűvel kezdjük. Összetett szó esetén a részeket is nagybetűvel írjuk
- Az objektumok nevét kis kezdőbetűvel írjuk . Összetett szó esetén a részeket nagybetűvel írjuk, továbbá az objektumok nevét aláhúzzuk.

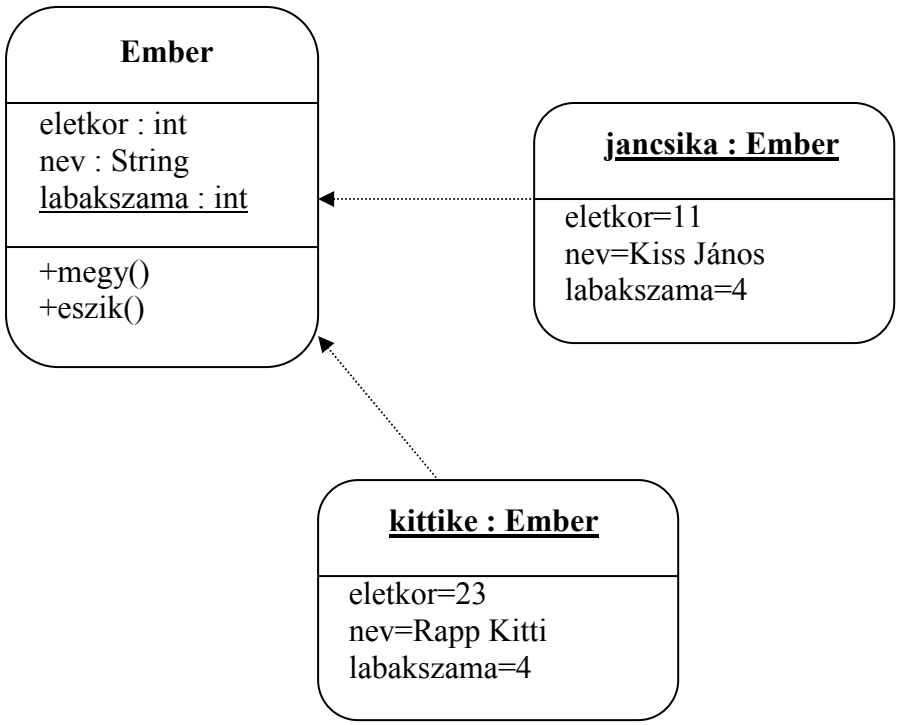
Az objektum és az osztály közötti kapcsolatot a következőképpen jelöljük:



Az ábrán láthatunk egy „Fák” osztályt és egy „egyTölgy” nevű objektumot. A szaggatott vonal jelzi, hogy utóbbi a Fák osztály egy példánya.

Azok kedvéért akiknek a „fás” példa nem szimpatikus, álljon itt még egy példa:
Ha körbenézünk a világban, rengeteg embert látunk körülöttünk. Nagyon jól tudjuk, hogy az ismert mondás teljesen igaz, miszerint „Nincs két egyforma ember”.
Ha most az a feladatunk, hogy állítsunk össze egy osztályt, akkor mindenkinek eszébe jut, hogy készítsünk egy Ember osztályt. Egy ember tulajdonságai közé tartozhat (a fához hasonlóan) az életkora, de nevet is rendelhetünk egy emberhez. Ha azt akarjuk megállapítani, hogy milyen metódusok rendelhetők az emberhez, akkor olyan igét kell keresnünk, ami az emberrel kapcsolatos. Ilyen viselkedés például az, hogy képesek vagyunk járni, és táplálkozni.

Az osztálydiagram tehát a következők szerint írható fel:



Természetesen egy osztálynak több példánya is lehet.

5. Az UML jelölési rendszer

A tantárgynak nem célja a teljes, szabványos UML jelölési rendszer bemutatása. Az UML-el részletesen foglalkozik az Információs rendszerek fejlesztése című tantárgy. A következőkben csak néhány fontosabb jelölési elemet foglalunk össze annak érdekében, hogy a feladatok megoldása során jó kiindulópont legyen egy UML diagram:

- Az osztályt nagy kezdőbetűvel jelöljük.
- Az objektumot kis kezdőbetűvel jelöljük, és aláhúzzuk (esetenként kettősponttal elválasztva megadjuk az osztály nevét is. Pl.: egyFa : Fak)
- Osztályváltozókat és osztálymetódusokat (statikus tagokat) aláhúzással jelöljük
- Hozzáférési módosítók jelölése:
  - public: +
  - private: -
  - protected: #
- Szokás még kettősponttal elválasztva megadni a változók típusát
- Metódusok esetében a paraméterlista is megadható

6. Osztályok használata a Java-ban

Az osztálydefiníció két részből áll:

- Változók deklarálása
- Metódusok leírása

A változók lehetnek példányváltozók és osztályváltozók. Példányváltozók esetén a változó értéke a példányosítás során inicializálódnak, értékük példányonként más és más lehet. Az osztályváltozók nem a példányokhoz tartoznak, hanem az osztályhoz. Az osztályváltozó értéke az összes példány számára ugyanaz. Az osztályváltozókat a static kulcsszóval lehet megkülönböztetni, UML-ben pedig aláhúzással jelöljük a fent említett módon.
Például egy Család osztályban a keresztnév példányváltozó, mert minden egyes családtagra nézve más és más (lehet) az értéke, de a családnevet érdemes osztályváltozóként definiálni, mert az mindenkinél ugyanazt az értéket jelenti.

A változók és függvények deklarációja előtt tehát módosítók állhatnak:

- **Hozzáférési módosítók**
  - *public*: mindenki hozzáférhet a változóhoz (+)
  - *private*: az adott osztályon kívül nem férhet hozzá senki (-)
  - *protected*: kizárólag ebből az osztályból és az osztály utódaiból lehet elérni (#)
- **Egyéb módosítók**
  - *static:* osztálytag (statikus tag)
  - *final:* végleges, nem módosítható

- *abstract:* üres metódus, ezt majd egy utódosztályban kell kifejteni

Az osztálydefiníció:

Osztályokat a class kulcsszóval tudunk definiálni:

```
modosito class Osztálynév
{
  // osztály törzs
}
```

A class kulcsszó előtt is szerepelhetnek módosítók:

- *public*: az adott osztály hozzáférhető más csomagokból is
- *abstract*: ezek az osztályok nem példányosíthatók
- *final*: nem terjeszthető ki

A különböző osztályokat célszerű külön fordítási egységben rögzíteni, és a fájl nevének kötelező az osztály nevét adni! Ha egy fordítási egységbe szeretnénk több osztályt definiálni, akkor figyelnünk kell arra, hogy csak egyetlen publikus osztály lehet (ez tartalmazza a main() osztálymetódust) és a fájl nevének meg kell egyeznie ezen publikus osztály nevével.

7. Objektumok használata a Java-ban

A korábbiakban láthattuk, hogy az objektumokat az osztályok példányosításával lehet létrehozni.

A példányosítás menete a következő:

```
Diak jancsika;
Jancsika = new Diak();
```

Ilyenkor a new operátor lefoglalja az objektum használatához szükséges tárterületet. A példányokon keresztül nyílik lehetőség az osztály változóinak és metódusainak eléréséhez a pont operátor használatával:

Példa:

```
String nev;
nev = jancsika.nev;
jancsika.megy(x,y);
```

8. Konstruktorok használata a Java-ban

A gyakorlaton bemutatott példákon tapasztalhattuk, hogy a példány változóit magunknak kellett inicializálni. Kívánatos lenne, ha már a példányosítás során lehetőségünk lenne a változók inicializálására. Ezt hivatott megoldani a konstruktor.

A konstruktor olyan programkód, amely a példányosításkor automatikusan végrehajtódik. Itt lesz lehetőségünk az objektum változóit beállítani illetve kedvünk szerint indíthatunk akár metódusokat is. A konstruktor nevének meg kell egyeznie az osztály nevével. A konstruktor a példányosítás során az osztály nevében átadott adatokat kapja paraméterül. Felhívom a figyelmet, hogy lehetőségünk van többféle szignatúrájú (paraméterezésű) konstruktor előállítására is.

Konstruktor készítése:

```
public class Diak
{
   String diaknev;
   int diakeletkor;

   public Diak(String nev, int eletkor)
   {                                          } az egyik konstruktor
      this.diaknev = nev;
      this.diakeletkor = eletkor;
   }

   public Diak(String nev)
   {                                          } a másik konstruktor
      this.diaknev = nev;
      this.eletkor = 20;
   }
}
```

A fenti kódrészlethez tartozó példányosítás a következőképpen működik:

```
Diak jancsika, pistike;
jancsika = new Diak(„Kiss János”, 18);      // példányosítás az első konstruktor használatával
pistike = new Diak („Nagy Pisti”);          // példányosítás a második konstruktor használatával
```

Biztosan sokakban felmerül a kérdés, hogy mi történt azokban a példákban, amikor még nem írtunk konstruktorokat. Abban az esetben automatikusan létrejött egy paraméterek nélküli konstruktor, ami a változóknak beállítja az alapértelmezés szerinti értéket. Miután létrehoztunk egy új konstruktort, a rendszer már nem foja létrehozni az automatikusat.

Megjegyzés:

A **this** kulcsszó egy objektumreferencia, ami az aktuális osztályra hivatkozik. (Az ősosztályra a **super** kulcsszóval hivatkozhatunk)