# 2.1.3. Objektum orientált programozás alapjai

## Bevezetés

Itt csak egy rövid összefoglalót találunk az objektum orientált programozásról. A objektum orientált programok konkrét megvalósítását lásd az adott programozási nyelvnél.

Az angol **Object-Oriented Programming** szavakból alkotott betűszó, röviden **OOP**. A valós világot úgy próbálja meg modellezni, hogy az sokkal közelebb álljon a valósághoz. A műveletek megalkotása helyett objektum hierarchiákat hozunk létre. A valós világból a számítógépen olyan olyan objektumokat hozunk létre, aminek tulajdonságai és viselkedéseik vannak.

## Osztály

Az osztály, angolul class. Egy egyed leírása, amiről információkat tárolunk. Ez az információ a programozási nyelvben változókat, és a változókon elvégzett műveletekből (eljárás, metódus) áll.

Felépíthetjük például a Dolgozók osztályát. A dolgozónak tárolhatjuk tulajdonságait. A tulajdonságok az osztály egy-egy változójában kapnak helyet. Például a dolgozó neve, kora, születésének ideje, fizetése, stb.

A metódusok, beállíthatják, lekérdezhetik a tulajdonságokat. Például a nevet, a születést, vagy meghatározhatják az osztály egyéb viselkedését.

Az osztályra úgy szokás tekinteni, mint az **objektum terve**. Még nem egy konkrét valamit ír le, csak egy terv.

## Mező

A osztályok és objektumok tulajdonságait néha mező néven említjük. Ezek az osztály változói. A mezők, a tulajdonságok, az attribútumok így ugyanazt jelölik.

A C# nyelv tartalmaz úgynevezett tulajdonság létrehozási lehetőséget, így ott a „tulajdonság” szónak más jelentése is van.

## Metódus

A mezőkön végrehajtható eljárások és függvények összefoglaló neve. A osztály viselkedésének is mondjuk.

## Osztály létrehozása

Osztályt általában a class kulcsszóval vezetjük be, utána pedig a nevét adjuk meg.

class Szemely {

[String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string) nev;

int kor;

}

[A diagram of a computer code

Description automatically generated](https://szit.hu/lib/exe/detail.php?id=oktatas%3Aprogramozas%3Aobjektum_orientalt_programozas&media=oktatas:programozas:class_pelda.png)

A mezők és a metódusok alkotják az osztály tagjait.

[A diagram of a computer code

Description automatically generated](https://szit.hu/lib/exe/detail.php?id=oktatas%3Aprogramozas%3Aobjektum_orientalt_programozas&media=oktatas:programozas:metodus_pelda.png)

[A yellow square with red circle and black text

Description automatically generated](https://szit.hu/lib/exe/detail.php?id=oktatas%3Aprogramozas%3Aobjektum_orientalt_programozas&media=oktatas:programozas:mezok_pelda.png)

### Python megközelítésben

[A computer code with text and arrows

Description automatically generated](https://szit.hu/lib/exe/detail.php?id=oktatas%3Aprogramozas%3Aobjektum_orientalt_programozas&media=oktatas:programozas:osztaly_letrehozasa_3.png)

## Konstruktor

A konstruktor egy speciális metódus. Speciális mivel lefut egy objektum létrehozása során.

Néhány nyelven a konstruktor neve megegyezik az osztály nevével. Ilyen a Java, C#. De a Python nyelvben a konstruktor neve init()

A konstruktornak nem jelölhetünk meg visszatérési értéket, mivel alapértelmezett feladat hogy visszaadja az osztály egy példányát.

Konstruktor Java és C# nyelven:

class Szemely {

public [String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string) nev;

public int kor;

public Szemely() {

this.nev = "névtelen";

this.kor = -1;

}

}

[A diagram of a function

Description automatically generated](https://szit.hu/lib/exe/detail.php?id=oktatas%3Aprogramozas%3Aobjektum_orientalt_programozas&media=oktatas:programozas:konstruktor_pelda.png)

A konstruktor mindig akkor fut le, amikor egy objektumot létrehozunk.

### Konstruktor Python módra

[A screen shot of a computer code

Description automatically generated](https://szit.hu/lib/exe/detail.php?id=oktatas%3Aprogramozas%3Aobjektum_orientalt_programozas&media=oktatas:programozas:osztaly_letrehozasa_konstruktor.png)

## Destruktor

Némely nyelvben van destruktor is. A destruktor akkor hajtódik végre, ha valamilyen objektumot megszüntetünk.

A Java nyelvben nem használunk destruktort.

A programozási nyelvekben általában a destruktor neve is megegyezik az osztály nevével, de egy ~ karaktert szokás elé tenni.

## Objektum

Az osztály egy példányosított változata.

A fenti Dolgozó osztályból ha egy példányt hozunk létre, az már egy konkrét dolgozó lesz. Lehet például egy „Nagy József” nevű dolgozó, aki 28 éves, 480000 - Ft a fizetése. Az objektumnak is adnunk kell egy nevet. Például Joska.

Példa Java és C# nyelven:

Dolgozo janos = new Dolgozo();

[A yellow rectangular sign with black text

Description automatically generated with medium confidence](https://szit.hu/lib/exe/detail.php?id=oktatas%3Aprogramozas%3Aobjektum_orientalt_programozas&media=oktatas:programozas:peldanyositas.png)

Az objektumot az osztályból hozzuk létre. Olyan ez mint egy változó létrehozása. Az egyenlőség jel jobboldalán a new kulcsszó való az objektum példány létrehozására. A new után mindig a konstruktort hívjuk.

## Elérés

Az osztály egyes tagjai (változók és metódusok) elérése változtatható.

* A **public** elérés, azt jelenti, hogy az adott tag más osztályokból is elérhető.
* A **private** elérés, azt jelenti, hogy más osztályokból nem érhető el a tag, még az örökölés lévén létrejött osztályokban sem.
* A **protected** elérés, az jelenti, hogy csak az örökölés során létrehozott osztályokban érhető el.

## Öröklés

Egy osztályból újabb osztályt hozhatunk létre, ennek folyamatát nevezzük öröklésnek. Az öröklés során az új osztály örökli az előző minden tagját, beleértve a mezőket és a metódusokat.

Öröklés C# nyelven:

class Szemely {

public String nev;

public int kor;

}

class Dolgozo : Szemely {

String munkakor;

}

## Többalakúság

A metódusok és a konstruktorok több alakban is meghívhatók. Az osztályok öröklés révén több formában is példányosíthatók. Ezeket a lehetőségeket nevezzük többalakúságnak.

### Többalakúság örökléssel

class Allat {

public void eszik() {

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.println("evés...");

}

}

class Macska extends Allat {

public void nyavog() {

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.println("miau");

}

}

public class Tobbalakusag\_01 {

public static void main([String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string)[] args) {

Macska macska = new Macska();

Allat allat;

allat = macska;

macska.eszik();

macska.nyavog();

allat.eszik();

/\* Az allat objektum a macska objektumra mutat,

mégsem lehet futtatni az allat objektumon

a nyavog() metódust. Mintha több alakja lenne.

\*/

}

}

### Többalakúság interfésszel

interface Allat {

public void beszel();

}

class Macska implements Allat {

public void beszel() {

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.println("miau");

}

}

class Kutya implements Allat {

public void beszel() {

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.println("vau");

}

}

public class Program01 {

public static void main([String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string)[] args) {

Allat a = new Macska();

Allat b = new Kutya();

a.beszel();

b.beszel();

}

}

### Függvény paraméter és öröklés

public class Program01 {

public static void main([String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string)[] args) {

Macska macska = new Macska();

Kutya kutya = new Kutya();

Allat allat = new Allat();

beszeltet(allat);

beszeltet(macska);

beszeltet(kutya);

}

public static void beszeltet(Allat allat) {

allat.beszel();

/\* Az allat objektum másként viselkedik

ha macska objektumot kap, és másként

ha kutya objektumot kap \*/

}

}

class Allat {

public void beszel() {

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.println("aaaaa");

}

}

class Macska extends Allat {

public void beszel() {

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.println("miau");

}

}

class Kutya extends Allat {

public void beszel() {

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.println("vau");

}

}

### Függvény paramétere és interfész

public class Program01 {

public static void main([String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string)[] args) {

Macska macska = new Macska();

Kutya kutya = new Kutya();

beszeltet(macska);

beszeltet(kutya);

}

public static void beszeltet(Allat allat) {

allat.beszel();

/\* Az allat objektum másként viselkedik

ha macska objektumot kap, és másként

ha kutya objektumot kap \*/

}

}

interface Allat {

public void beszel();

}

class Macska implements Allat {

public void beszel() {

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.println("miau");

}

}

class Kutya implements Allat {

public void beszel() {

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.println("vau");

}

}

## Virtuális metódus

Egy metódus akkor virtuális, ha az örökölt osztályokban felülírható. A programozási nyelvekben ezt az ősosztályban kell beállítani, általában a virtual kulcsszóval. A Java nyelvben nem szükséges ezt megtenni, mivel minden metódus alapértelmezettként felülírható, vagyis virtuális.