# I. Objektumorientált programozás

## 1. Az objektumorientált programozás alapjai

### Objektumok és objektumosztályok

A számítógépes programok közvetve vagy közvetlenül a körülöttünk lévő világ elemeihez illeszkednek, gyakorlati feladatokat oldanak meg. Ilyen feladatnak tekinthetjük az adatbázis-kezelésen, a mérnöki-tudományos számításokon túl akár egy videó lejátszását vagy játékprogramok futtatását is. Ezért célszerű olyan programozási modellt alkalmazni, amely illeszkedik a környező világ építőköveihez, leírja tulajdonságaikat és viselkedésüket. Ezt az elvet valósítja meg az **objektumorientált programozás**, röviden **OOP**. Az OOP olyan módszert nyújt a programozók számára, amely lehetővé teszi a programok bonyolultságának csökkentését, a megbízhatóság és a hatékonyság növelését.

Az OOP objektumokból építi fel a programot.

> **Objektum:** a valós világ elemeinek programozási modellje. Az objektum adatokat tárol, és kérésre tevékenységeket végez.

Lássunk példát objektumokra! Az iskolában diákok tanulnak. Egy diák adataihoz tartozik a neve, születési ideje, lakcíme. A diák a tanórán felel, hiányzást igazol, tornázik. A diák objektummodelljének – többek között – ezeket az adatokat kell tartalmaznia, ezeket a tevékenységeket végeznie.

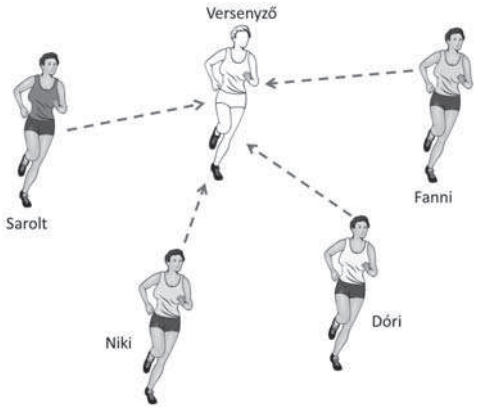
A grafikus felhasználói felület programozásakor is objektummal dolgozunk. Megadjuk például egy parancsgomb helyzetét, méretét, feliratát. Elkészítjük eseménykezelő eljárásait. A *Programozási ismeretek* tankönyv bemutatja a grafikus felhasználói felület számos objektumát.

Egy osztályba sok diák jár, a programablakban több parancsgombot helyezhetünk el. Programjainkban gyakran sok objektum tárol ugyanolyan adatokat, és végez ugyanolyan tevékenységeket. Az egymáshoz hasonló objektumokat osztályokba soroljuk.

> **Objektumosztály:** az egymáshoz hasonló objektumok általánosítása, gyűjtőfogalma. Az objektumosztály határozza meg a hozzá tartozó objektumok típusát.

Az objektum által tárolt adatokat szokás az objektum **mezőinek,** adattagjainak vagy tagváltozóinak hívni. Programjainkban az *osztályszintű* változók jelentik az objektumok mezőit. Az objektumokat függvények vagy eljárások hívásával „kérjük meg" a tevékenységek végzésére. Ezek az alprogramok az objektum **metódusai**. A metódusokat gyakran tagfüggvényeknek, illetve eljárástagoknak nevezzük. Az objektum tehát a hozzá tartozó mezők és metódusok összessége. A mezők és metódusok együttesen alkotják az objektum tagjait. A metódusok hívása helyett azt is mondhatjuk, hogy **üzenetet küldünk** az objektumnak, amely a metódus végrehajtásával válaszol az üzenetre.

A szakirodalomban néha csak azokat a változókat nevezik mezőknek, illetve azokat az alprogramokat hívják metódusoknak, melyek az osztályszintűnél nagyobb hatókörrel rendelkeznek (kívülről is elérhetők). Könyvünkben követjük a Visual Studio szóhasználatát, mely szerint mezőnek tekintjük az osztályszinten (tehát az eljárásokon/függvényeken kívül) deklarált változókat, illetve metódusnak minden, az osztályban definiált alprogramot.




*Az objektumok*
*az objektumosztály konkrét egyedei*

Minden egyes objektum egy-egy jól meghatározott osztályhoz tartozik. Az osztály forráskódjában definiáljuk a hozzá tartozó objektumok mezőit és metódusait.

Az egyes objektumok a megfelelő objektumosztály **példányai**. A diák objektumosztály egy példánya (objektuma) Kovács István vagy Nagy Irén. A programablakban a parancsgomb (*Button*) objektumosztály példányait helyezzük el (*Button1*, *Button2* stb.).

Az objektumok és objektumosztályok használata megkönnyíti az összetett, bonyolult programok elkészítését, mert összefogja, egységbe zárja az összetartozó elemeket.

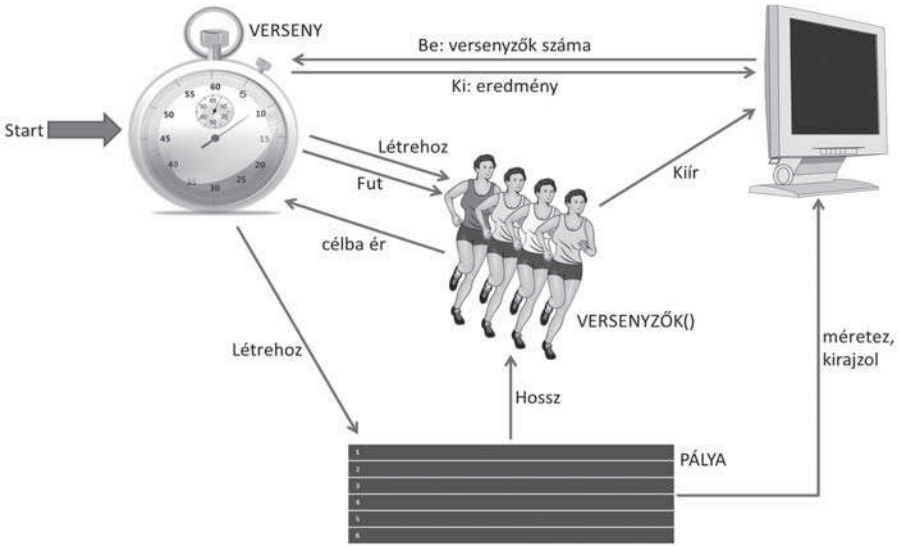
**Egységbezárás** (encapsulation): az összetartozó adatokat és a hozzájuk kapcsolódó tevékenységeket egyetlen objektumban fogjuk össze.

**1. gyakorlat.** Keressünk az interneten objektumorientált programozási nyelveket! Telepítsük egy ilyen nyelv fejlesztőrendszerét! A továbbiakban szükségünk lesz rá.

## Futóverseny

A következő leckékben egy futóverseny szimulációjával mutatjuk be az objektumorientált programozás alapjait. A program kirajzolja a képernyőre a pályát, amelyen különböző színű, sorszámokkal jelképezett versenyzők futnak végig, balról jobbra. A sötét háttér miatt csak világos színeket választunk. Futás közben a versenyzők helyzetét egy ciklussal változtatjuk. A ciklusban minden versenyző véletlenszerű mértékben lép előre. A verseny addig tart, amíg egy léptetés során egy vagy több versenyző be nem ér a célba. Őket mind győztesnek tekintjük, tehát megengedjük a holtversenyt is. A verseny végén kiírjuk a győztes versenyző(k) sorszámát. A működő programot a forrásfájlok között található *Futóverseny* mutatja be.


*A futóverseny képernyőképe*




*A futóverseny objektumai és feladataik*
*A programban szereplő mezőket és metódusokat nagy kezdőbetűvel írjuk*

Az OOP szemléletmódjának megfelelően osszuk fel a végrehajtáshoz szükséges feladatokat kis egységekre, azaz alakítsuk ki a megvalósításhoz szükséges objektumokat! Közben arra törekszünk, hogy **minden objektum lehetőleg önállóan végezze el a feladatát, a lehető legkisebb mértékben támaszkodjon más objektumokra**. Nyilván szükségünk lesz versenyzőobjektumokra, amik a versenyzőket jelképezik. Kialakítunk továbbá egy versenyobjektumot, amely a verseny indítását, adminisztrálását végzi. Végül célszerű létrehozni egy pályaobjektumot is, amely a pálya kirajzolásáról, adatainak (hosszának) tárolásáról gondoskodik.

*A futóverseny objektumai, a mezők és metódusok megadásával:*

- Pályaobjektum: a versenypályát jelképezi.

  *Hossz* mező: a pálya hossza. A versenyzők használják fel a cél elérésének ellenőrzéséhez.
  *Létrehoz* metódus: a versenyzők számának megfelelően méretezi az ablakot, és kirajzolja a képernyőre a célvonalat. Az ablak szélessége alapján meghatározza a *Hossz* értékét. A versenyzők számát argumentumként adjuk meg.

- Versenyzőobjektum: egy versenyző adatait és metódusait tartalmazza.

  Mezők:
  *Mez*: a versenyző mezének sorszáma.
  *Szín*: a versenyző mezének színe.
  *Helyzet*: a versenyző pozíciója a pályán.
  *Színek()*: a lehetséges színeket tartalmazó tömb.
  *Vélszám*: véletlenszám-objektum a szín választásához, illetve a helyzet változtatásához.
  Metódusok:
  *Létrehoz*: inicializálja a mezőket. A metódus argumentuma a versenyző mezének sorszáma. A színt véletlenszerűen választja ki.

*Fut*: lépteti a versenyzőt a verseny során. Véletlenszerűen választott mértékben megnöveli a *Helyzet* mező értékét. Ha a versenyző beér a célba, akkor hozzáadja a mez sorszámát a versenyobjektum győzteseket tartalmazó halmazához.
*Kiír*: megjeleníti a képernyőn a versenyző helyzetét. A *Fut* metódus hívja.

- Versenyobjektum: elindítja és vezérli a versenyt.

  Mezők:
  *Pálya*: pályaobjektum.
  *Győztesek*: a győztesek mezének sorszámát tartalmazó halmaz.
  *VersenyzőkSzáma*: a versenyzők száma.
  *Versenyzők()*: a versenyzőobjektumokat tartalmazó tömb.
  Metódusok:
  *Start*: megkérdezi a felhasználótól a versenyzők számát, majd létrehozza a versenyzőobjektumokat, és meghívja *Létrehoz* metódusukat. Létrehozza a pályaobjektumot, amelynek meghívja a *Létrehoz* metódusát. A *Futás* metódus hívásával elindítja a versenyt, a végén pedig az *Eredmény* metódussal kiíratja az eredményt.
  *Futás*: magát a versenyt jelképezi. Ciklussal lépteti a versenyzőket mindaddig, amíg valamelyikük be nem ér a célba (egy cikluson belül esetleg egyszerre több is).
  *Eredmény*: a verseny végén kilistázza a győztes versenyzőket. Ehhez a *Győztesek* halmazt használja fel, melybe a célba érő versenyzők beírták mezük sorszámát.
  *Vár*: a megadott időtartamra leállítja a végrehajtást, egyébként a versenyt nem tudnánk követni a képernyőn. A *Futás* metódus hívja.

A szerepek kiosztása természetesen másként is elvégezhető. Például kialakíthattunk volna külön objektumot a győztes versenyzők nyilvántartására és kiírására, a versenyzők helyett a versenyobjektum adhatná a célba beérők sorszámát a győztesek halmazához, a versenyzők léptetését végezhetnék időzítőobjektumokkal stb.

A programot konzolalkalmazásként készítjük el, így a grafikus felhasználói felület elemei nem zavarják az objektumok áttekintését. A konzolalkalmazás indítja el a futóversenyt úgy, hogy a *Main* eljárás létrehozza a versenyobjektumot, majd meghívja a *Start* metódust.

A futóverseny programját fokozatosan építjük fel. A közben felmerülő problémák megoldásához egyre újabb OOP-eszközöket ismerünk meg.

**2. gyakorlat.** Írjuk meg a futóverseny programját OOP-eszközök nélkül!

## Ábrázolás osztálydiagramon

Programunkban osztályokat definiálunk a tervben szereplő objektumok alapján. Az osztályok szerkezetét úgynevezett osztálydiagrammal tesszük áttekinthetővé. Az osztálydiagram fontos eleme az objektumorientált programok tervezését segítő UML-nek (Unified Modeling Language: egységes modellező nyelv).

Az **osztálydiagram** téglalapban tünteti fel az osztály nevét, mezőit és metódusait. A mezőket és metódusokat vízszintes vonallal választjuk el egymástól.

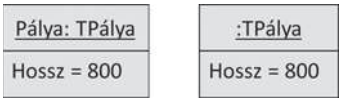
**3. gyakorlat.** A futóverseny leírása alapján jelöljük be az alábbi osztálydiagramra, hogy egy-egy osztály objektumai mely másik osztály mezőit kérdezik le vagy módosítják, illetve mely metódusokat hívják! Melyik osztály objektuma hoz létre egy másik osztályhoz tartozó objektumot vagy objektumokat?

A mellékelt osztálydiagramon csak az általunk definiált osztályokat tüntettük fel. Nem ábrázoltuk külön a fejlesztőrendszerben rendelkezésünkre álló osztályokat (véletlenszám, halmaz, tömb).

Az osztálydiagramok mellett gyakran használják az objektumdiagramot is. Az **objektumdiagram** az objektumokat téglalapokban ábrázolja, feltüntetve az objektum nevét, osztályát, továbbá mezőinek értékét. Az objektum nevét, osztályát aláhúzzuk, és kettősponttal választjuk el egymástól. Ha nem lényeges az objektum neve, akkor el is hagyhatjuk.

Az UML-re a későbbiekben még visszatérünk, kibővítjük az osztály- és objektumdiagram fogalmát.


*A futóverseny osztálydiagramja*
*Az osztályok nevének kezdőbetűje (T) jelzi, hogy új típust definiálunk.*




*A pályaobjektum kétféle ábrázolási módja*

## Objektumosztály definiálása, objektumok létrehozása

Objektumosztályt az *OSZTÁLY … OSZTÁLY VÉGE* kulcsszavak között definiálunk a forráskódban. A definícióban a mezők (változók) deklarációi és metódusok (alprogramok) definíciói szerepelnek a szokásos módon:

```
OSZTÁLY Osztálynév
    változók (mezők) deklarálása
    eljárások, függvények (metódusok) definiálása
OSZTÁLY VÉGE
```

Mivel az osztály a hozzá tartozó objektumok típusa, ezért könyvünkben az osztályok nevét *T* betűvel kezdjük. A *TPálya* osztály definíciója például (vázlatosan):

```
OSZTÁLY TPálya
    VÁLTOZÓ Hossz MINT Egész
    ELJÁRÁS Létrehoz(VersenyzőkSzáma MINT Egész)
        az ablak méretének módosítása
        a célvonal kirajzolása
    ELJÁRÁS VÉGE
OSZTÁLY VÉGE
```

Az objektumosztályokat definiálhatjuk a *Module1.vb*, illetve *Form1.vb* kódfájlban (a *Module1* modul, illetve a *Form1* osztály definícióján kívül), de célszerű inkább külön kódfájlt hozzáadni a projekthez (*Project/Add New Item*, *Class*). Egy kódfájlban több osztálydefiníció is szerepelhet, azonban áttekinthetőbb programot kapunk, ha minden egyes kódfájl csak egyetlen osztály definícióját tartalmazza. A *View/Other Windows/Class View* paranccsal megnyitott *Class View* munkaablakban könnyen áttekinthetjük az osztályokat (az ősosztályokkal együtt – lásd később). A *Class View* munkaablakot csak a *Professional* változatban találjuk meg.


*Class View a Visual Basic-ben*

**4. gyakorlat.** Hozzunk létre új konzolalkalmazást! Adjuk hozzá a *TPálya* kódfájlt, amelyben definiáljuk a *TPálya* osztályt! A futóverseny leírásának megfelelően deklaráljuk a *Hossz* mezőt, majd készítsük el a *Létrehoz* metódus kódját! Ha a fejlesztőrendszer esetleg *Public* megjelöléssel látja el az egyes mezőket vagy metódusokat, akkor ezzel most ne törődjünk!

Az objektumokat – egyelőre – a konzolalkalmazás moduljának (*Module1*) *Main* metódusában hozzuk létre, a már jól ismert módon. Először deklaráljuk az objektumra hivatkozó, az osztálynak megfelelő típusú változót, majd az *Új* operátorral elkészítjük az objektumot, és hozzárendeljük a változóhoz:

*Objektumnév* MINT *Objektumosztály*
*Objektumnév* = Új *Objektumosztály*

A pályaobjektum létrehozása például:

```
VÁLTOZÓ Pálya MINT TPálya
Pálya = Új TPálya
```

Vegyük figyelembe, hogy az *Objektumnév* itt egy hivatkozás típusú változó, amely az objektumnak a memóriában elfoglalt helyét mutatja (lásd például a *Programozási ismeretek* tankönyvet). Az ilyen változót a továbbiakban **objektumváltozónak** nevezzük. Az objektumváltozó deklarálásakor még nem jön létre maga az objektum, majd csak az *Új* operátor készíti el.

Egy objektum létrehozását az objektumosztály **példányosításának** szokás nevezni.

A példányosítás után ne felejtsük el meghívni a *Létrehoz* metódust, amely elvégzi az ilyenkor szükséges tevékenységeket. A metódust – szabály szerint – az objektum nevével minősítjük:

```
Pálya.Létrehoz(VersenyzőkSzáma)
```

Az objektumok létrehozásának és inicializálásának ez a szétválasztása sok hibalehetőséget rejt magában, például könnyen megfeledkezhetünk róla. Hamarosan megismerkedünk az inicializálást automatikusan elvégző metódus készítésének módjával.

**5. gyakorlat.** Írjuk meg a pályaobjektumot létrehozó és a *Létrehoz* metódust meghívó kódot! Futtassuk az alkalmazást!

Programozási összefoglaló: Objektumok és objektumosztályok