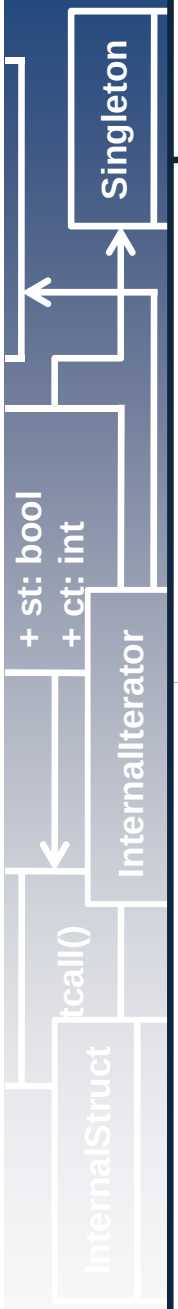
ELTE-IK
Programozási technológia 1.

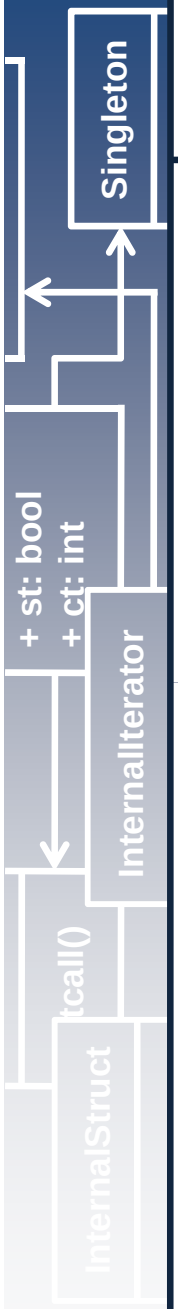
2. gyakorlat:
Programozási paradigmák, az objektumorientált programozás

© 2008.09.17. Giachetta Roberto
groberto@inf.elte.hu
http://people.inf.elte.hu/groberto



Programozási paradigmák

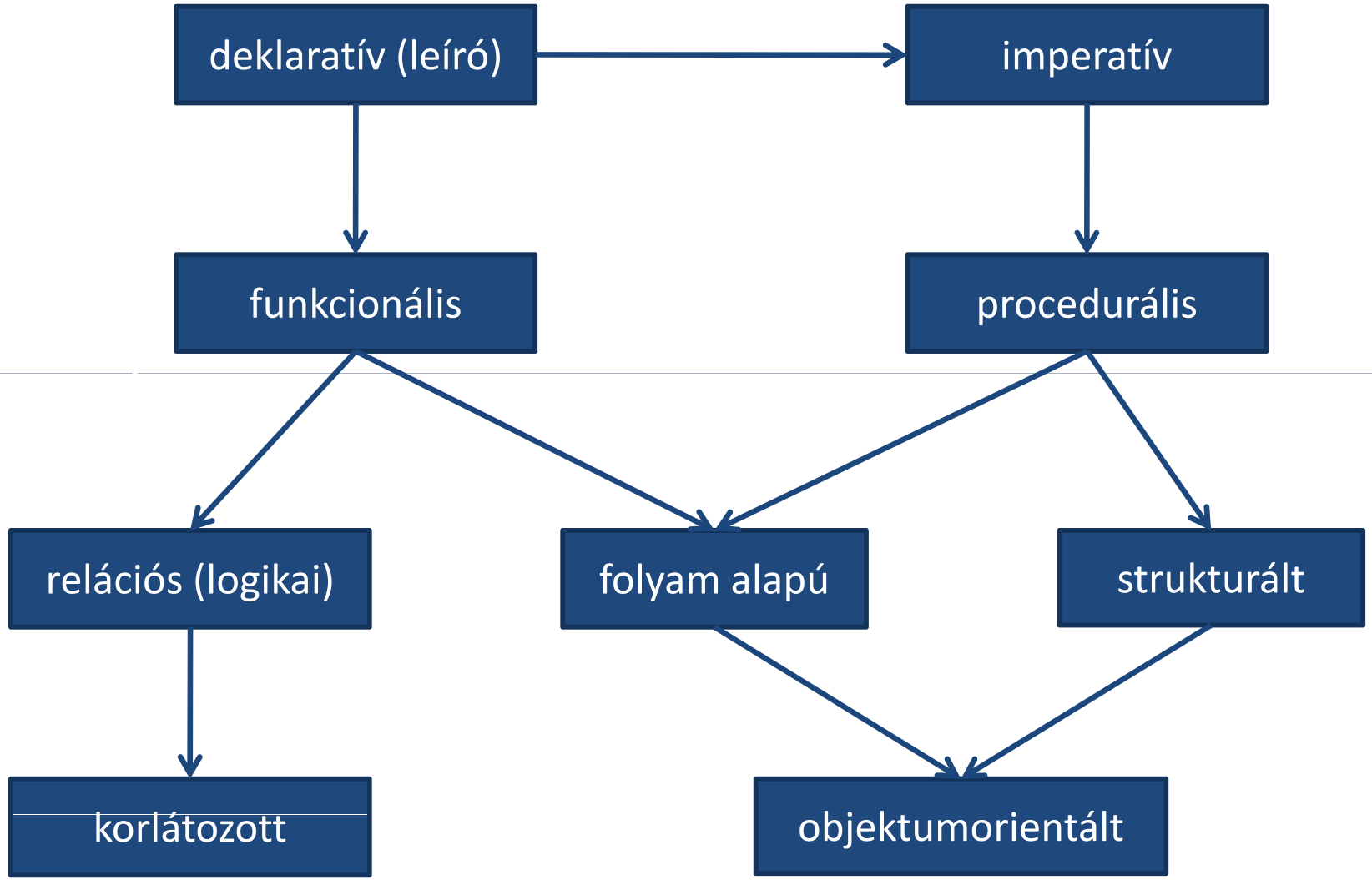
- A szoftverfejlesztés során a legfőbb cél, hogy a kész programrendszer megfeleljen a funkcionális és minőségi követelményeknek
- Emellett, a fejlesztők számára fontos, hogy a kész szoftver fejlesztése a lehető legoptimálisabb legyen
  - a szoftverfejlesztési modell meghatározza a fejlesztés módját és menetét
  - azt nem, hogy milyen legyen a fejlesztés stílusa, milyen absztrakciós szinten történjen a megvalósítás, hiszen ezek a tényezők nem határozzák meg a szoftvert önmagát, csak a fejlesztés folyamatát
- A szoftverek tervezésének és programozásának módszerét nevezzük **programozási paradigmá**nak

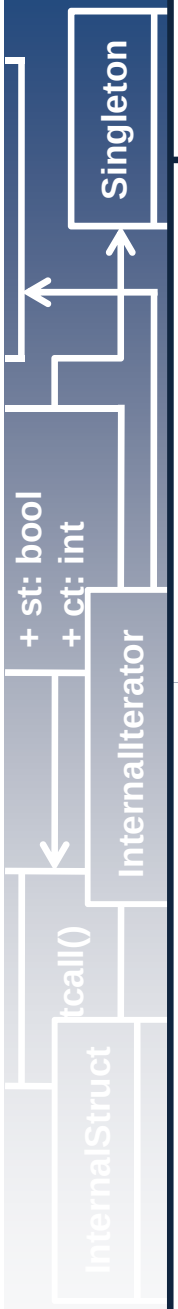


Programozási paradigmák

- A paradigma meghatározza a programozás stílusát és az absztrakciós szintet
- Vannak általános célú, és szűk körben alkalmazott paradigmák, minden adott feladatra megtalálható a legjobban alkalmazható paradigma, vagy paradigmák
- Általában az elemzési szakasz végén célszerű meghatározni az alkalmazni kívánt paradigmákat (egy projekten belül akár programegységenként alkalmazhatunk más paradigmát)
  - a paradigma meghatározza az alkalmazható programozási nyelvek körét is, és fordítva
  - egyes programozási nyelvek több paradigmát is támogatnak, így a programozási nyelv szempontjából is meghatározhatjuk

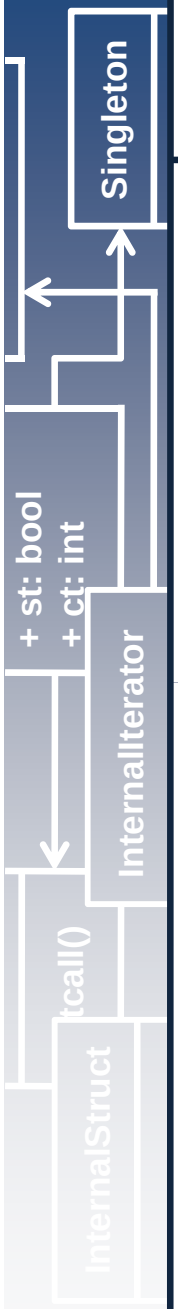
Elsődleges programozási paradigmák





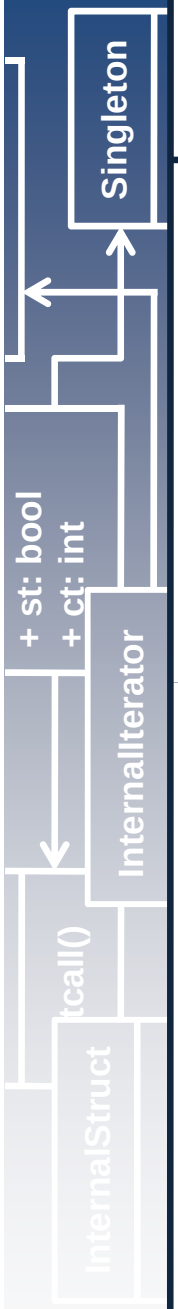
Elsődleges programozási paradigmák

- **Deklaratív:**
  - a program a tartalmát, megjelenését írja le, nem pedig a funkció módját
  - általában nem alkalmas teljes értékű programok írására, mert nem Turing teljes (azaz nem képes minden feladatot elvégezni)
  - megszokott alkalmazási területe konfigurációs fájlok, interfész felületek programozására, ezért rendszerint kiegészítésként szolgál más paradigma használata mellett
  - deklaratív nyelvek: XSLT (XML), HTML, XAML, SQL
  - egyes funkcionális nyelvek is támogatják deklaratív részek írását (pl. LISP, Erlang, Prolog), illetve imperatív nyelvekben is előfordulhatnak deklaratív részek (pl. LINQ)



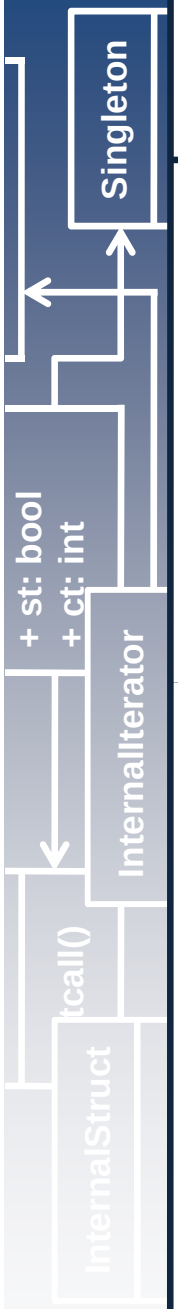
Elsődleges programozási paradigmák

- **Funkcionális:**
  - a programfutás matematikai függvények kiértékelésének sorozata, alapja a lambda-kalkulus
  - nincsenek benne állapotok és változó adatok, gyakori a rekurzió, könnyű a párhuzamosítás
  - korábban csak kutatásokban alkalmazták, mostanság megjelenik a piaci szoftverekben is
  - a nagyobb absztrakciós szint miatt magasabb az erőforrásigény, de rövidebb a programkód
  - funkcionális nyelvek: LISP, Erlang, Clean, Haskell, R, Mathematica
  - egyes imperatív nyelvekben lehet szimulálni függvénymutatók és lambda-függvények segítségével



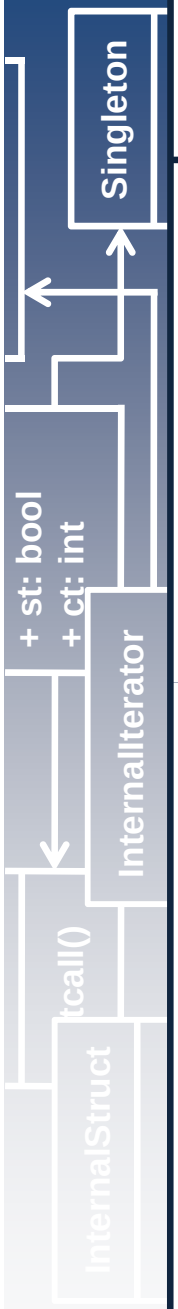
Elsődleges programozási paradigmák

- **Relációs, logikai:**
  - a program logikai kiértékelések sorozata, általában implikációkon keresztül
  - a programozó feladat, hogy a program „igaz legyen”, és megfelelően hatékony
  - logikai nyelvek: Prolog, Oz, Castor for C++
- **Korlátozott:**
  - nem a megoldás módját, hanem a megoldandó probléma tulajdonságait írja le, a változók kapcsolatát korlátozásokon keresztül adja meg
  - korlátozott nyelvek: B-Prolog, CHIP, Claire
  - imperatív nyelvek külön programkönyvtárakkal támogatják: Choco (Java), Disolver (C++)



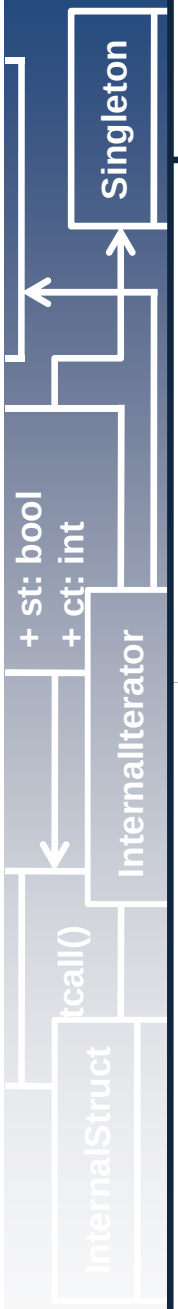
Elsődleges programozási paradigmák

- **Folyam alapú:**
  - a program fekete doboz jellegű alkalmazások hálózata, amelyek üzenetküldéssel kommunikálnak egymással
  - az egyes folyamatok párhuzamosan, aszinkron módon futnak, és információs csomagokat fogadnak, illetve küldenek
  - nyelvek: Linda, FBP
- **Imperatív:**
  - a program utasítások sorozata, amelyek állapotváltozásokat okoznak, így a program az állapotok mentén írható le
  - a feladat megvalósításának lépéseit írja le, azaz milyen akciók sorozatát kell végrehajtani, ezért általában kizárja a programkódban való ugrások (GOTO) lehetőségeit



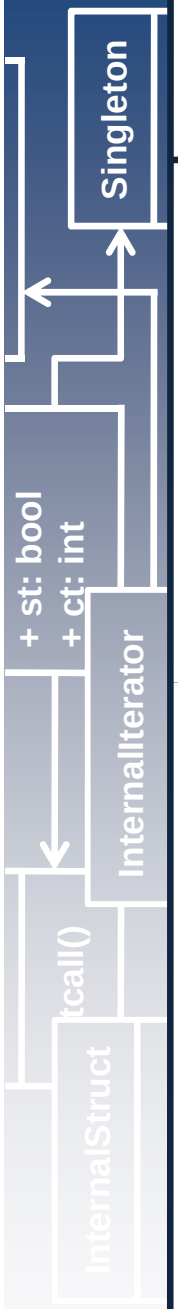
Elsődleges programozási paradigmák

- **Procedurális:**
  - az imperatív elv továbbfejlesztése, az utasítások eljárásokba vannak csoportosítva (a program modularizálva van), és ezen eljárások összessége oldja meg a feladatot
  - a procedurális megközelítés előnyei:
    - kód újrahasznosítás
    - programkód könnyű nyomon követése, gyors áttekinthetősége
    - a programszerkezet struktúrájának tetszőleges kialakítása
  - procedurális nyelvek: Fortran, C, Ada, BASIC, Pascal, PHP, Python, ECMAScript



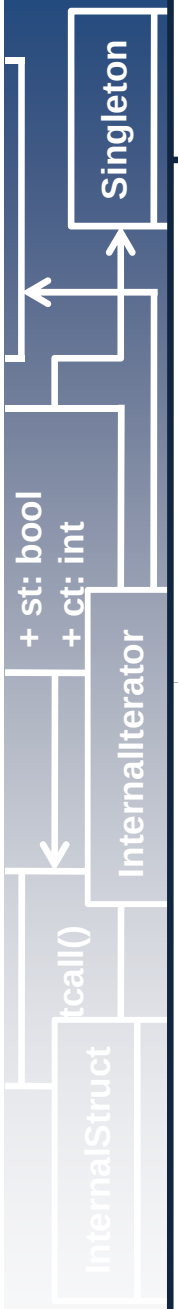
Elsődleges programozási paradigmák

- **Strukturált:**
  - a procedurális programozás egy változata
  - a programok olyan részegységekre tagolódik, egy belépési ponttal, és egy kilépési ponttal
  - tervezésnél felülről lefelé megközelítés, azaz a fő problémát bontjuk részekre, és azokat további részekre, satöbbi
  - a programban használt adatstruktúrák a programegységeknek megfelelően strukturálódnak
- **Objektum-orientált:**
  - a programegységet egymással kommunikáló objektumok összessége adja, amelyek valamilyen relációban állnak egymással



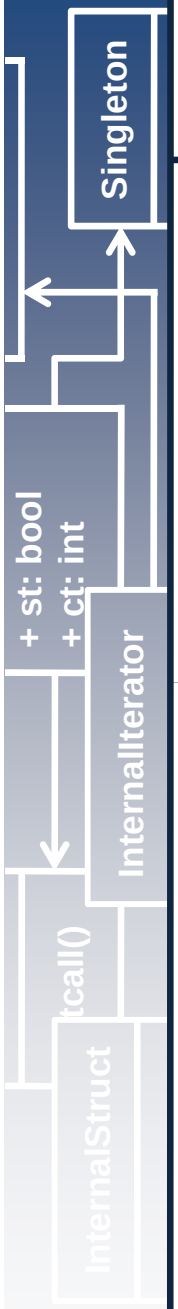
Programozási paradigmák

- Vannak úgynevezett másodlagos programozási paradigmák, amelyek a programozást stílusát nagyban befolyásolják, pl.:
  - eseményvezérelt: a programfutás mentetét az interfészeken keresztül érkező események vezérlik
  - folyamat alapú: a programban egymással párhuzamosan futó folyamatok kommunikálnak
  - iteratív: a feladatot iteratívan oldja meg, rekurzív hívás nélkül
  - rekurzív: a feladatot rekurzív hívások sorozatával oldja meg
  - sablon programozás: az adatszerkezetek sablonokon keresztül állnak össze
  - meta programozás: a folyamatot a program metainformációs (programmanipulációs) szinten vezérli



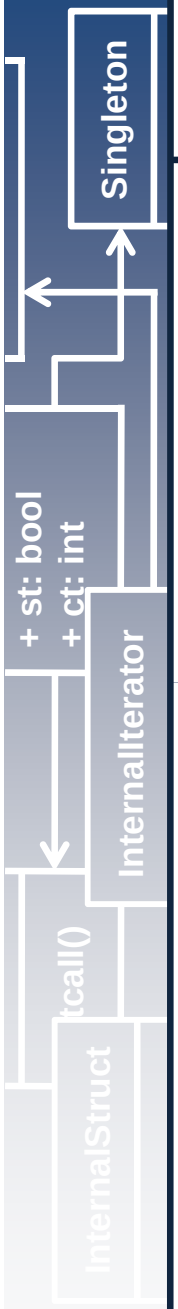
Objektumorientáció

- Az objektumorientált programozásban a feladatot nem felülről lefelé, hanem alulról felfelé oldjuk meg, azaz azonosítjuk azon alkotóelemeket, amelyek a végrehajtásban részt vesznek, és ezek kapcsolatának, kommunikációjának módját tárjuk fel
  - az alkotóelemek külön önálló egységeket alkotnak, amelyek adott feladatok elvégzésére alkalmasak, és belső változókkal és műveletekkel rendelkeznek, amelyekkel különböző állapotokat vehetnek fel
  - a kommunikáció üzenetküldéseken keresztül valósul meg
  - általában nagyobb feladatok megoldására alkalmazható, amelyek procedurális módon túl komplikáltnak bizonyulnak



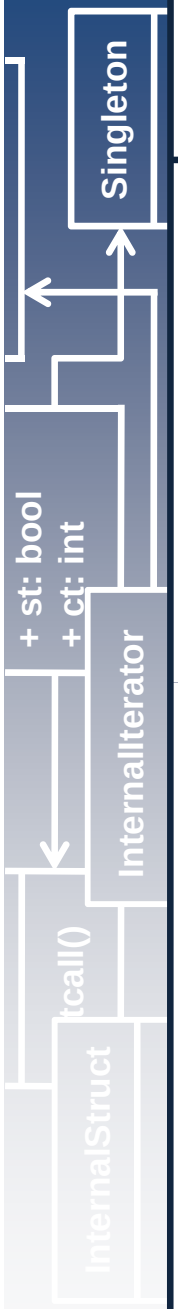
Objektumorientáció

- A való világ modellezéséből indul ki
- Először a 60-as években tűnt fel a Simula programozási nyelvben az objektumorientált támogatás, majd a 80-as években megjelent az első teljesen objektumorientált nyelv, a SmallTalk
- Manapság a legnépszerűbb programozási paradigma, a programozási nyelvek jelentős része támogatja
- Objektumorientált támogatással rendelkező nyelvek: C++, Object-C, Ada, Object Pascal (Delphi), Common LISP, F#, Visual Basic, PHP, ECMAScript, Python, Perl, D, Visual Prolog, …
- Objektumorientált nyelvek: Smalltalk, JAVA, C#, Eiffel, Ruby, …



Objektumorientáció alkotóelemei

- **Osztály (Class):**
  - az alkotóelemek absztrakt jellemzőit, karakterisztikáját, azaz típusát adja meg (tekinthető egy tervrajznak is, amely alapján a példányokat le lehet gyártani)
  - megadja, milyen részekből, és értékekből állhat egy alkotóelem, és milyen tevékenységre képes (milyen műveleteket tud futtatni)
  - tekinthető változók (konstansok) és eljárások (függvények) gyűjteményének
  - az osztály változóit **attribútumoknak**, az eljárásokat pedig **metódusoknak** nevezzük
  - az osztály biztosítja a program modularitását



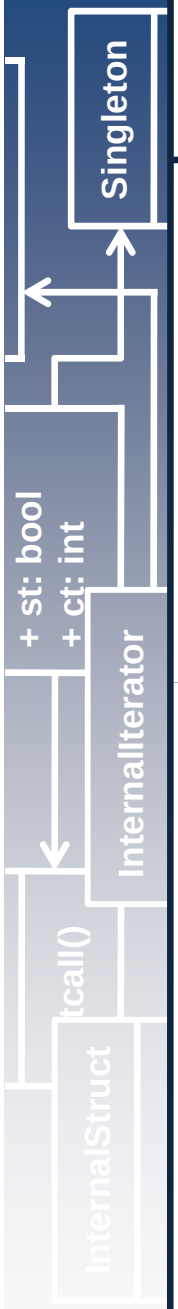
Objektumorientáció alkotóelemei

- az osztálynak kell biztosítania egy külső felületet (export interface), amely a külvilág (a többi objektum) számára látható, és amelyen keresztül a kommunikáció lezajlik, az osztály többi részét el kell rejteni a külvilágtól
- pl. kutya, villamos, autó

- **Objektum (Object):**
  - az osztály egy példánya, amely konkrét értékekkel bír, és meghatározza az egyedet
  - ténylegesen ilyen jönnek létre és működnek a program futása során, és mindegyikük önálló állapotokkal rendelkezik
  - egy objektumokhoz tetszőleges számú további objektum is kapcsolódhat
  - pl. Lassie, 2025-ös Combino, AII-911-es rendszámú autó

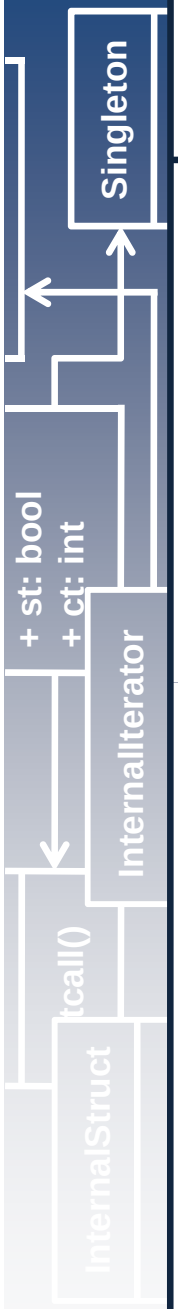
Objektumorientáció alkotóelemei

- **Öröklődés:**
  - az osztályoknál megszabhatunk általánosabb, illetve speciálisabb osztályokat, ahol a speciálisabbak rendelkeznek minden tulajdonsággal, amivel az általánosabb változat
  - pl.:



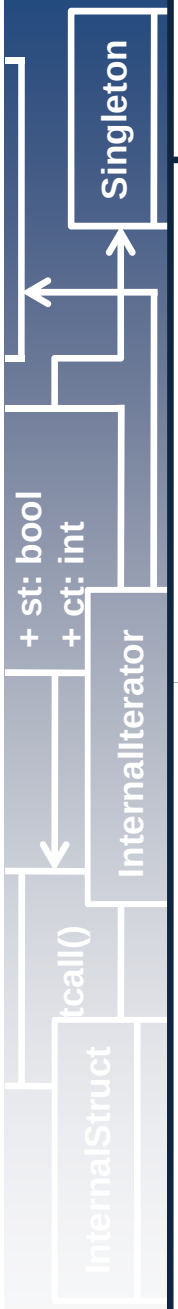
Objektumorientáció alkotóelemei

- ezt a folyamatot, amikor a speciálisabb átveszi az általános jellemzőit és működését, öröklődésnek nevezzük
- az öröklődés két irányát specializációnak, illetve általánosításnak nevezzük, az általánosabb objektumot ősnek, a speciálisabbat leszármazottnak nevezzük (ha csak egy szint a különbség, akkor szülőnek, illetve gyereknek)
- általában egy szülőnek több gyereke is van, ám csak egyes esetekben engedélyezett, hogy egy gyereknek több szülője is legyen (ezt többszörös öröklődésnek nevezzük)
  - többszörös öröklődés során a gyerek megkaphatja minden szülője minden tulajdonságát, illetve szabályozhatjuk, hogy melyik őstől melyeket kapja



Objektumorientáció alkotóelemei

- **Absztrakció:**
  - az objektumok tetszőleges reprezentálása lehetővé teszi, hogy az adott problémát több szinten kezeljük, azaz az összes lehetséges tulajdonságának egy tetszőleges halmazát válasszuk ki, és azokat használjuk az ábrázolásban
  - a reprezentációs szint megválasztása jelenti az absztrakció szintet
  - a számunkra lényeges tulajdonságokat megtartjuk, a lényegteleneket elhagyjuk
  - pl.: az autót ábrázolhatjuk rendszámmal, sebességgel és tömeggel, vagy ábrázolhatjuk, mint motor és karosszéria összessége, vagy akár minden egyes alkatrészét külön-külön vehetünk



Objektumorientáció alkotóelemei

- **Polimorfizmus:**
  - ha egy objektum olyan osztálynak példánya, amelynek több őse is van, akkor az az objektum tekinthető bármely ősének példányaként is
  - ezáltal lehetővé válik, hogy egy objektumnak több osztálya is legyen, és aktuálisan azt az osztályt használjuk, amely az aktuális környezetbe beleillik
  - pl.: ha egy raktérben vegyesen akarunk autókat és motorokat elhelyezni, akkor azt mondjuk, hogy egységesek személygépjárműveket helyezünk el (amelynek mindkettő speciális esete)
  - pl.: ha egy koordináta rendszerben köröket és négyzeteket is akarunk rajzolni, akkor egységesen geometriai alakzatokat rajzolunk