Programozási nyelvek és módszerek

BEVEZETÉS

Fontos

Oktató

- Tornai Kálmán
- 232-es szoba
- tornai.kalman@itk.ppke.hu

Tárgyadatok

- Programozási nyelvek és módszerek
- P-ITSZT-0007
- P-ITSZT-0045
- Hétfő: 10.15-13.00
  - Neumann ea.
  - Zárva van órákon kívül, kint kell várakozni.

Követelmények

- Jegy: Szóbeli vizsga
  - Vizsga előfeltétele az érvényes (>1) gyakorlati jegy a Java programozási nyelv tárgyból!
- Az előadások látogatása legalább 80%-ban kötelező
  - Maximum két hiányzás!
- A félév során két hétfő oktatási szünet
  - Tavaszi szünet

Irodalom

Kötelező irodalom (részek)

- Sebesta, Robert W., Concepts of programming languages, Edinburgh, Pearson, 2013, ISBN 978-0-273-76910-1
- Nyékyné Gaizler Judit et al., (szerk.), Programozási nyelvek, Budapest, Kiskapu, 2003

Ajánlott irodalom

- SCOTT, Michael L., Programming Language Pragmatics, Amsterdam, Morgan Kaufmann, 2009
- Comparative programming languages, Harlow, Addison-Wesley, 2001

Tematika

- Bevezetés
- A programozási nyelvek elemei
- Beépített adattípusok, változók, kifejezések
- Utasítások
- Alprogramok és paramétereik
- Absztrakt adattípusok
- Sablonok
- Kivételek
- Objektum-orientált programozás
- Helyesség
- Párhuzamosság
- A könyvtártervezés elvei
- A funkcionális programozás alapjai

A félév célja

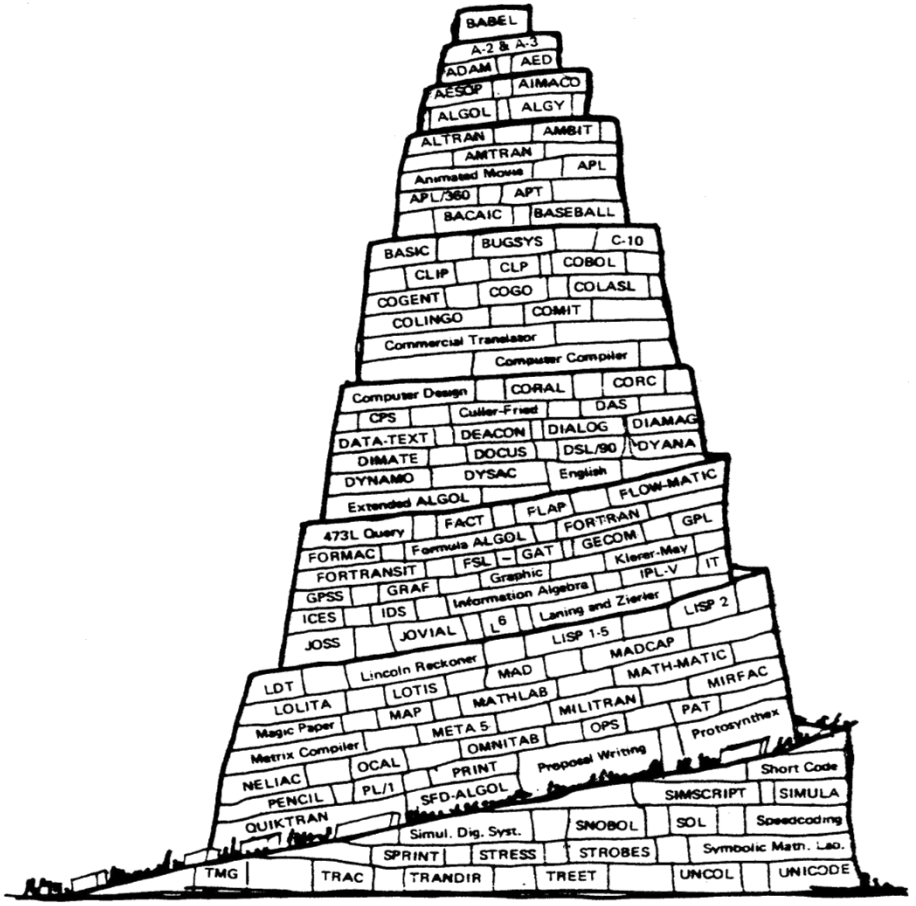
Programozási módszertan alapjai

Programozási nyelvek eszközkészlete

- Melyik nyelvet válasszuk egy feladatra?
- Milyen lehetőségek vannak?
- Milyen nyelvi stílusok vannak?
  - Ne ragadjunk bele a C++/Java logikájába
- Alapok későbbi programozási paradigmák, nyelvek elsajátítására
  - Mik a népszerű nyelvek most?
  - Mik lesznek?
    - Van még bőven idő a nyugdíjig …

Ráhangolódás

A nyelvek – Bábel tornya



Tower of Babel

Mother Tongues

Tracing the roots of computer languages through the ages

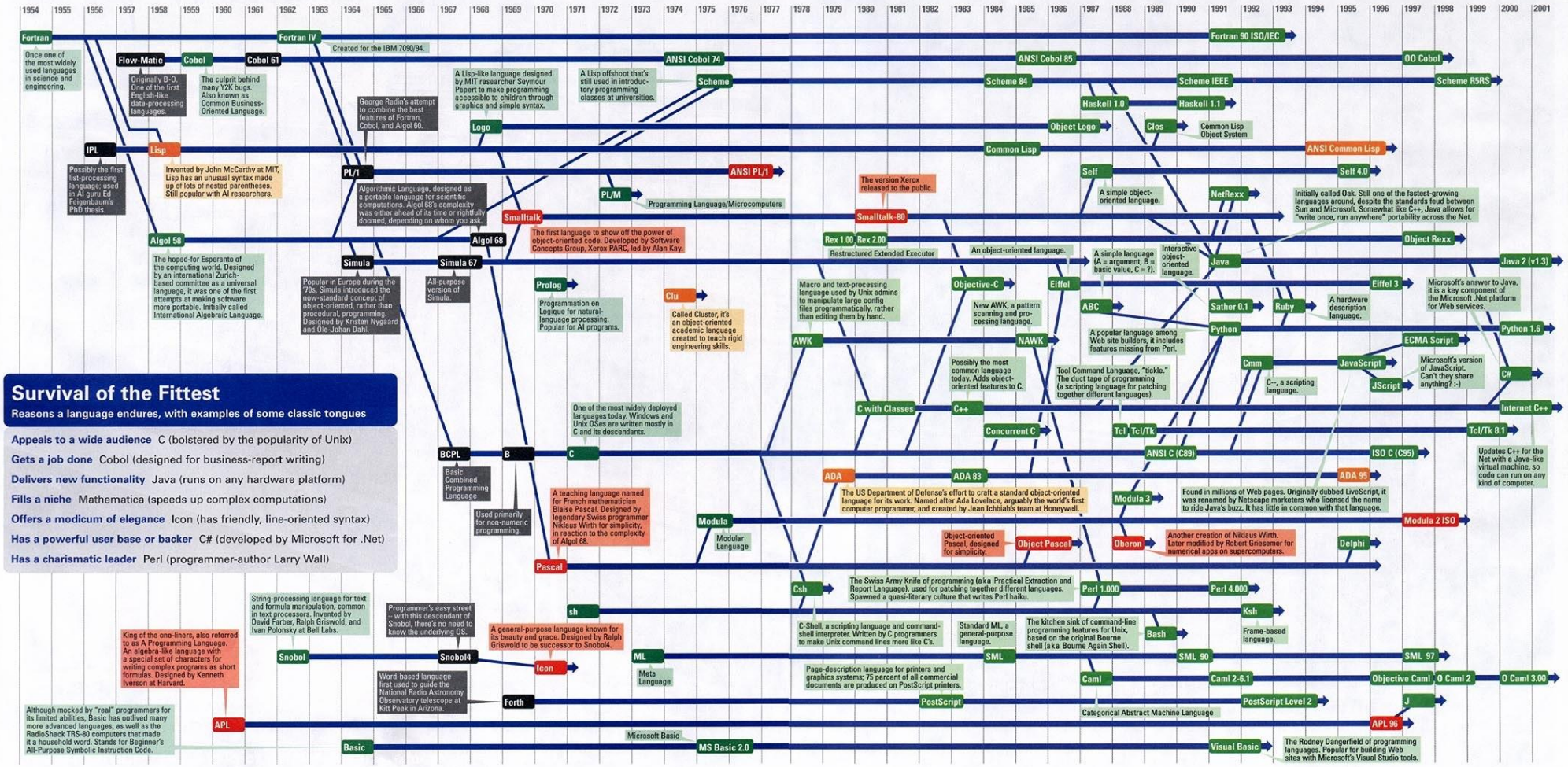
Just like half of the world's spoken tongues, most of the 2,300-plus computer programming languages are either endangered or extinct. As powerhouses C/C++, Visual Basic, Cobol, Java, and other modern source codes dominate our systems, hundreds of older languages are running out of life.

An ad hoc collection of engineers – electronic lexicographers, if you will – aim to save, or at least document, the lingo of classic software. They're combing the globe's 9 million developers in search of coders still fluent in these nearly forgotten lingua francas. Among the most endangered are Ada, APL, B (the predecessor of C), Lisp, Oberon, Smalltalk, and Simula.

Code-raker Grady Booch, Rational Software's chief scientist, is working with the Computer History Museum in Silicon Valley to record and, in some cases, maintain languages by writing new compilers so our ever-changing hardware can grok the code. Why bother? "They tell us about the state of software practice, the minds of their inventors, and the technical, social, and economic forces that shaped history at the time," Booch explains. "They'll provide the raw material for software archaeologists, historians, and developers to learn what worked, what was brilliant, and what was an utter failure." Here's a peek at the strongest branches of programming's family tree. For a nearly exhaustive rundown, check out the Language List at *www.informatik.uni-freiburg.de/Java/misc/lang_list.html*. **– Michael Menduno**

Key

- 1954 **Year Introduced**
- **Active:** thousands of users
- **Protected:** taught at universities; compilers available
- **Endangered:** usage dropping off
- **Extinct:** no known active users or up-to-date compilers
- **Lineage continues**



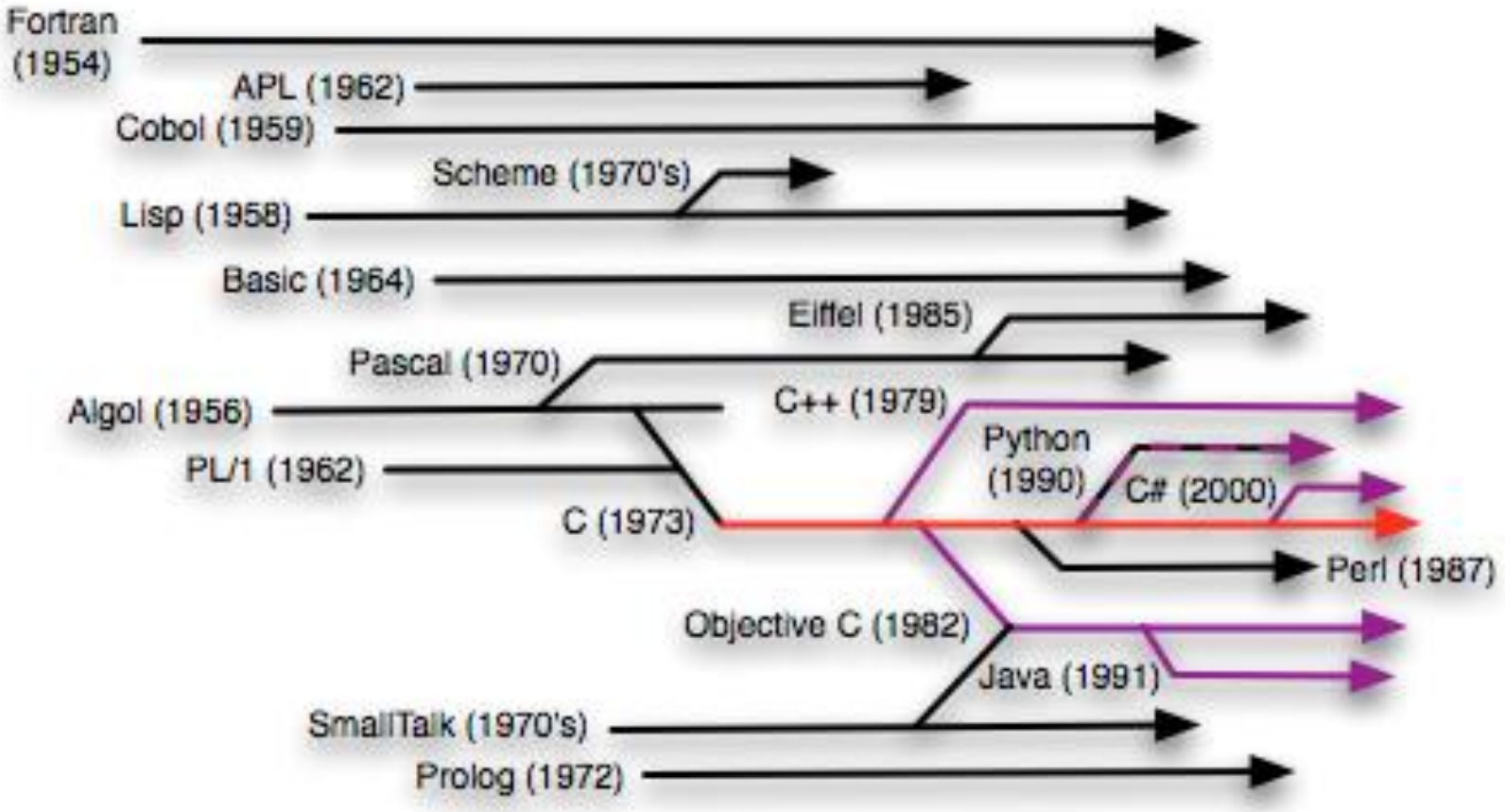
Survival of the Fittest

Reasons a language endures, with examples of some classic tongues

- **Appeals to a wide audience** C (bolstered by the popularity of Unix)
- **Gets a job done** Cobol (designed for business-report writing)
- **Delivers new functionality** Java (runs on any hardware platform)
- **Fills a niche** Mathematica (speeds up complex computations)
- **Offers a modicum of elegance** Icon (has friendly, line-oriented syntax)
- **Has a powerful user base or backer** C# (developed by Microsoft for .Net)
- **Has a charismatic leader** Perl (programmer-author Larry Wall)

Sources: Paul Boutin; Brent Hailpern, associate director of computer science at IBM Research; The Retrocomputing Museum; Todd Proebsting, senior researcher at Microsoft; Gio Wiederhold, computer scientist, Stanford University

„Fontosabb” nyelvek családfája



Miért van ennyi?

Számos ok

- Evolúció
  - Újabb nyelvi konstrukciók, megközelítések
  - Pl: OOP, szkriptnyelvek
- Speciális célok
  - C: alacsonyszintű programozás
  - Awk: karakter alapú feldolgozás
  - Python: könnyű prototipizálás, számítások
- Személyes preferenciák
  - Rekurziós algoritmusok <-> Iteratív algoritmusok
  - Pointerek használata <-> automatikus (implicit) referencing

Mitől sikeres egy nyelv?

Több ok

- Kifejező erő
  - Gyakori érv bizonyos nyelvek mellett / ellen, hogy „erős”
  - Matematikai értelemben nem igaz, mindegyik nyelv azonos kifejezőerővel bír, minden algoritmus leírható benne
    - Lásd később
    - A különbség a szintaktikai támogatottság
      - Le lehet írni, más kérdés, hogy mennyire bonyolultan
- Kezdők számára könnyű használat
- Könnyű implementálhatóság
  - Fordítóprogram, környezet
- Szabványosság
  - „Nyelvjárásoktól mentes”

Mitől sikeres egy nyelv?

Több ok

- Nyílt forrás
  - Manapság szinte bármely nyelvre igaz, hogy van nyílt forrású implementáció
- Jó fordítóprogram
  - Hibamentes
  - Jó optimalizálás
    - Gyors kódot állít elő
  - Könnyű használhatóság
    - Toolchain

Történelmi mérföldkövek

FORTRAN (1950-es évek közepe-vége)

- hangsúly a tudományos programozáson

LISP (1950-es évek vége)

- rekurzió, dinamikus helyfoglalás és felszabadítás

ALGOL 60 (1960)

- az adattípus fogalma
- érték- és név szerinti paraméterátadás

COBOL (1960)

- angol-szerű szintaxis

Történelmi mérföldkövek

BASIC (1960-as évek közepe)
- Interpretált

PL/I (1960-as évek közepe)
- Általános célú
- Párhuzamosan végrehajtódó taszkok
- Kezeli a futási idejű kivételeket
- Pointerek mint adattípusok

APL (1960 körül)
- Tömb és mátrix kezelés

SNOBOL (1960-as évek közepe)
- Szövegkezelés, szövegminták hasonlítására

Történelmi mérföldkövek

SIMULA 67 (1967)
- Osztály fogalmának bevezetése – egységbezárás
- Öröklődés

ALGOL 68 (1968)
- A legtöbb ezt követő programozás nyelv merített a tervezéséből

Pascal (1970-es évek eleje)
- Programozás oktatásra

C (1970-es évek eleje)
- rendszerprogramozásra

Prolog (1970-es évek közepe)
- Logikai nyelv

Történelmi mérföldkövek

Ada (1970-es évek vége)

- A Pentagon megbízásából
- Kivételkezelés
- Sablonok
- Párhuzamosság támogatása (taszkok, randevúk)
- Helyességellenőrzés (2012-től)

Smalltalk (1980)

- Objektumorientált – máshogy

C++ (1980-as évek eleje óta)

- SIMULA 67 és a Smalltalk + C

Történelmi mérföldkövek

Objective C (1980-as évek eleje)
- C és Smalltalk

Eiffel (1980-as évek eleje)
- Objektumorientált
- Tervezés szerződéssel a függvényhívó és a függvény között

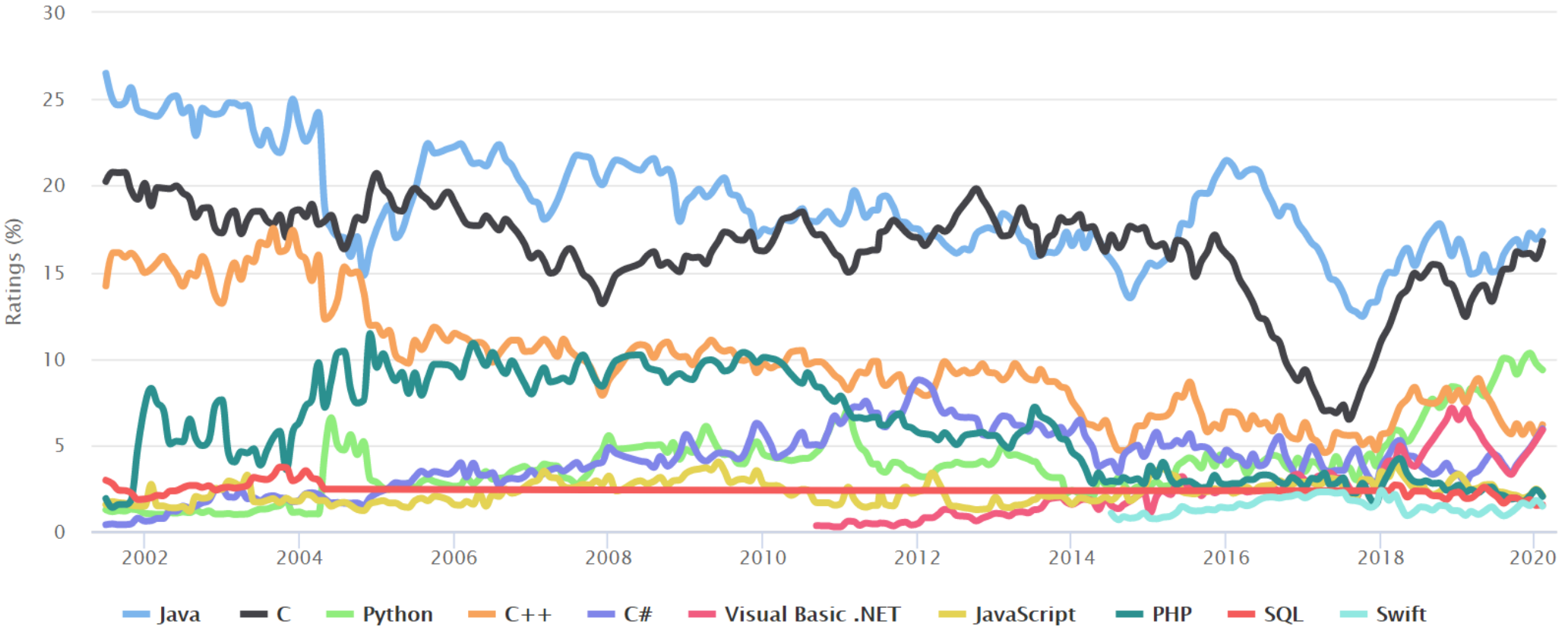
Java (1990-es évek közepe)
- OOP
- Párhuzamosság támogatása
- GC
- Kivételkezelés

C#
- „Microsoft Java-ja”
- .NET alapú fejlesztés támogatása

Mitől fontos egy programozási nyelv?

HTTPS://WWW.TIOBE.COM/TIOBE-INDEX/
HTTPS://SPECTRUM.IEEE.ORG/COMPUTING/SOFTWARE/THE-TOP-PROGRAMMING-LANGUAGES-2019

TIOBE Index


Ratings (%)
30
25
20
15
10
5
0
2002
2004
2006
2008
2010
2012
2014
2016
2018
2020
Java
C
Python
C++
C#
Visual Basic .NET
JavaScript
PHP
SQL
Swift

TIOBE Index

2020	2019	2018	2017	2016	2015	2014	2013	2012	2011	2010	2009	2008	Nyelv
1	1	1	1	1	2	2	1	2	1	1	1	1	Java
2	2	2	2	2	1	1	2	1	2	2	2	2	C
3	3	4	5	5	8	8	7	8	8	7	8	6	Python
4	4	3	3	3	3	4	4	4	3	3	4	3	C++
5	7	5	4	4	5	5	5	5	4	6	7	8	C#
6	5	7	6	7	9	10	12	15	39				Visual Basic .NET
7	6	6	7	9	6	9	11	11	10	11	9	9	JavaScript
8	8	9	10	6	7	6	6	6	5	4	3	5	PHP
9	9												SQL
10	15	12	14	16									SWIFT

TIOBE Index

2020	2019	2018	2017	2016	2015	2014	2013	2012	2011	2010	2009	2008	Nyelv
11	16	19	13										Go
12	14	15	9	13									Assembly
13	12	8	16	17	18	44							R
14	23										12		D
15	18	11	12	11	20	12	10	10	11	10	10	11	Ruby
16	11	18	19	18	17	14	18	20	22				MATLAB
17	19	20	20	19	13	15	22	19	17	20	14	13	PL/SQL
18	17	13	11	10	11	20	15	12	12	12	11	10	Delphi / Object Pascal
19	13	10	8	11	12	13	9	9	9	9	6	7	Perl
20	10	16	18	14	4	3	3	3	6	8	18	42	Objective-C

TIOBE hosszútávú összehasonlítás

Nyelv	2020	2015	2010	2005	2000	1995	1990	1985
Java	1	2	1	2	3	-	-	-
C	2	1	2	1	1	2	1	1
Python	3	7	6	6	22	21	-	-
C++	4	4	4	3	2	1	2	12
C#	5	5	5	8	8	-	-	-
Visual Basic .NET	6	10	-	-	-	-	-	-
JavaScript	7	8	8	9	6	-	-	-
PHP	8	6	3	4	27	-	-	-
SQL	9	-	-	97	-	-	-	-
Objective-C	10	3	21	37	-	-	-	-
Lisp	31	18	16	13	14	5	3	2
Ada	35	29	24	15	15	6	4	3
Pascal	229	16	13	65	11	3	15	5

IEEE Rank – 2019

Rank	Language	Type	Score
1	Python	web, desktop, embedded	100.0
2	Java	web, mobile, desktop	96.3
3	C	mobile, desktop, embedded	94.4
4	C++	mobile, desktop, embedded	87.5
5	R	desktop	81.5
6	JavaScript	web	79.4
7	C#	web, mobile, desktop, embedded	74.5
8	Matlab	desktop	70.6
9	Swift	mobile, desktop	69.1
10	Go	web, desktop	68.0

Miért tanuljunk (tovább) programozási nyelveket?

Egyben a tárgy célja is

- Megérteni a nyelvekben rejlő további funkcionalitást
  - C++ union típuskonstrukció
  - Változó számú argumentumok
- Alternatív módon kifejezni megoldásokat
  - Választott nyelvnek megfelelően
    - Matlab: mátrix műveletek
    - Python: list comprehensing
      - `x = [2 ** x for x in range(10) if x > 5]`
- Hibakeresők, assemblerek, …
  - Használatuk, céljuk megértése

Miért tanuljunk (tovább) programozási nyelveket?

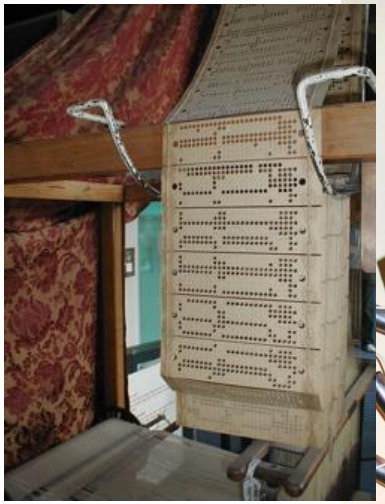
Egyben a tárgy célja is

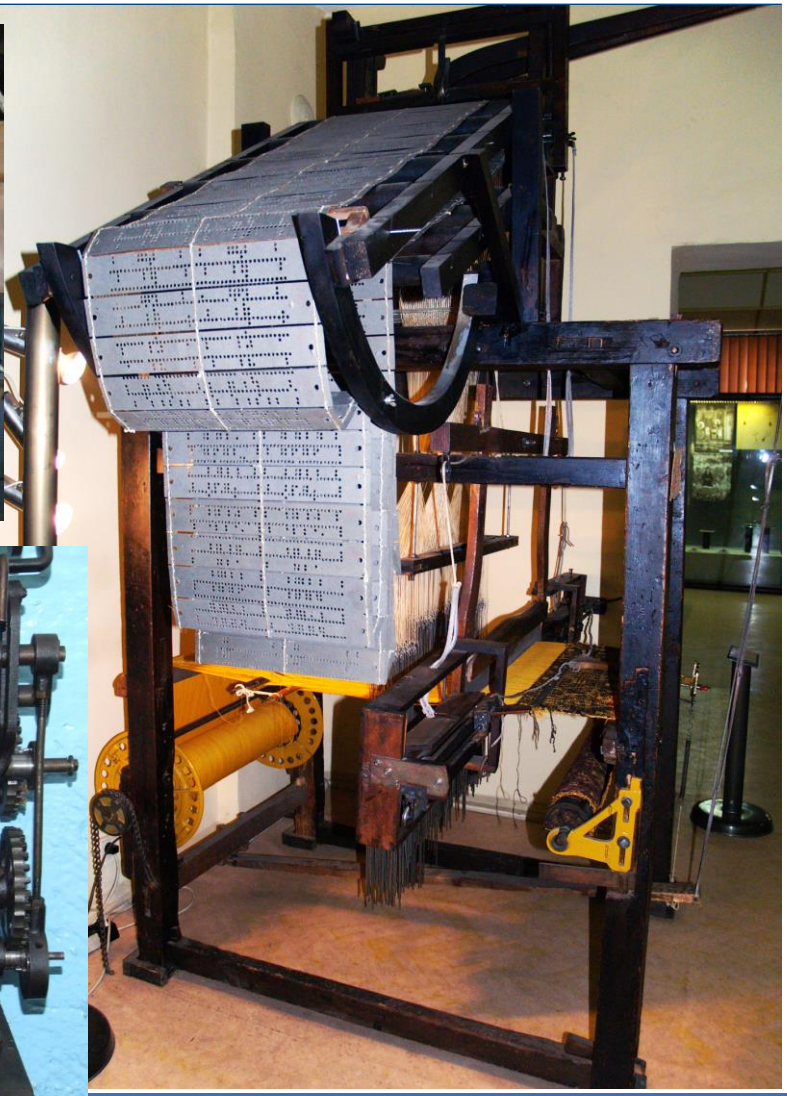
- Hatékony, hasznos nyelvi eszközök „átültetése”
  - Megvalósítás / Szimulálás más nyelven
- Szélesíteni a látókört a programnyelvek nyelvi eszközkészletét illetően
  - További konstrukciók
  - Megfelelő eszköz kiválasztása
  - Nem megrettenni egy új nyelvtől
    - Stratégia a megtanulásra

Szoftverminőség

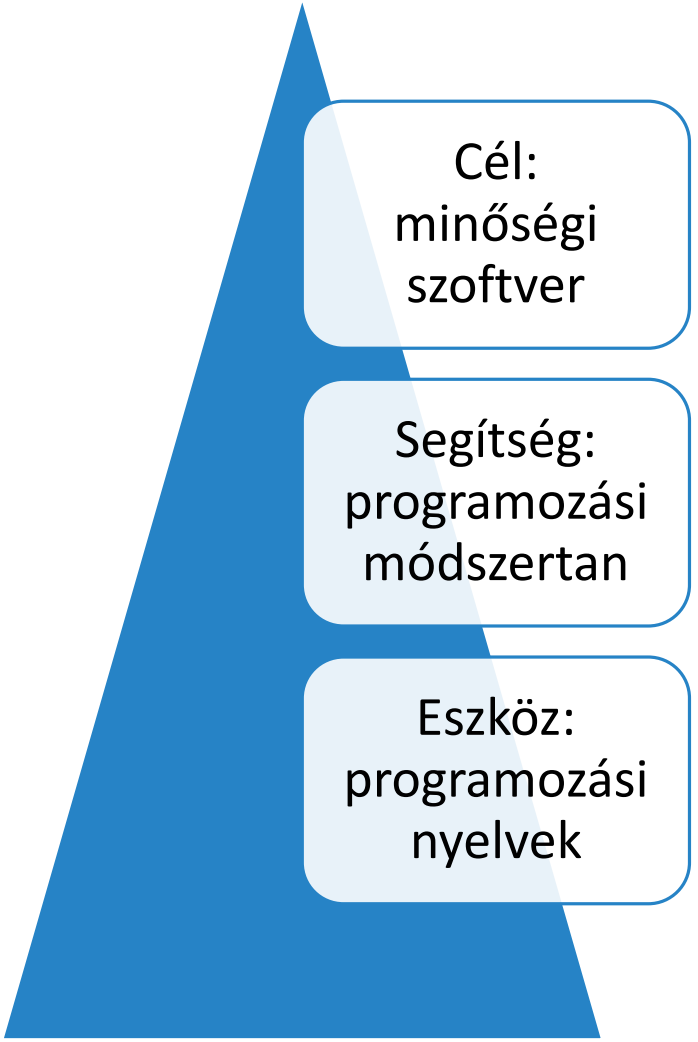
Miért készítünk programokat?

Joseph Marie Jacquard
(szövőgép)





Minőségi szoftver



Szempontok

- Helyesség
- Megbízhatóság
- Karbantarthatóság
- Újrafelhasználhatóság
- Kompatibilitás
- Hordozhatóság
- Hatékonyság
- stb.

Minőségi szoftver

Helyesség

- A program helyes, ha pontosan megoldja a feladatot, megfelel a kívánt specifikációnak.
  - Ehhez szükséges a pontos és minél teljesebb specifikáció.

Megbízhatóság

- A program megbízható, ha
  - *Helyes*, és
  - Abnormális helyzetekben sem történik katasztrófa
    - Valamilyen „ésszerű" működést produkál
    - Abnormális: a specifikációban nem leírt

Minőségi szoftver

Karbantarthatóság

- A karbantarthatóság annak mérőszáma, hogy milyen könnyű a programterméket a *specifikáció változtatásához* adaptálni.
  - Egyes felmérések szerint a szoftver költségek 70 %-át szoftver karbantartásra fordítják!
- Növelésének két alapelve
  - Tervezési egyszerűség
  - Decentralizáció – minél önállóbb modulok létrehozása

Újrafelhasználhatóság

- A szoftver termék, vagy komponens egy másik, új alkalmazásban felhasználható.
  - Ez vonatkozik a szoftvertermék
    - Egészére
    - Részére

Minőségi szoftver

Kompatibilitás

- A kompatibilitás megmutatja, hogy mennyire könnyű a szoftver termékeket egymással kombinálni.
  - Nem légüres térben fejlesztünk!

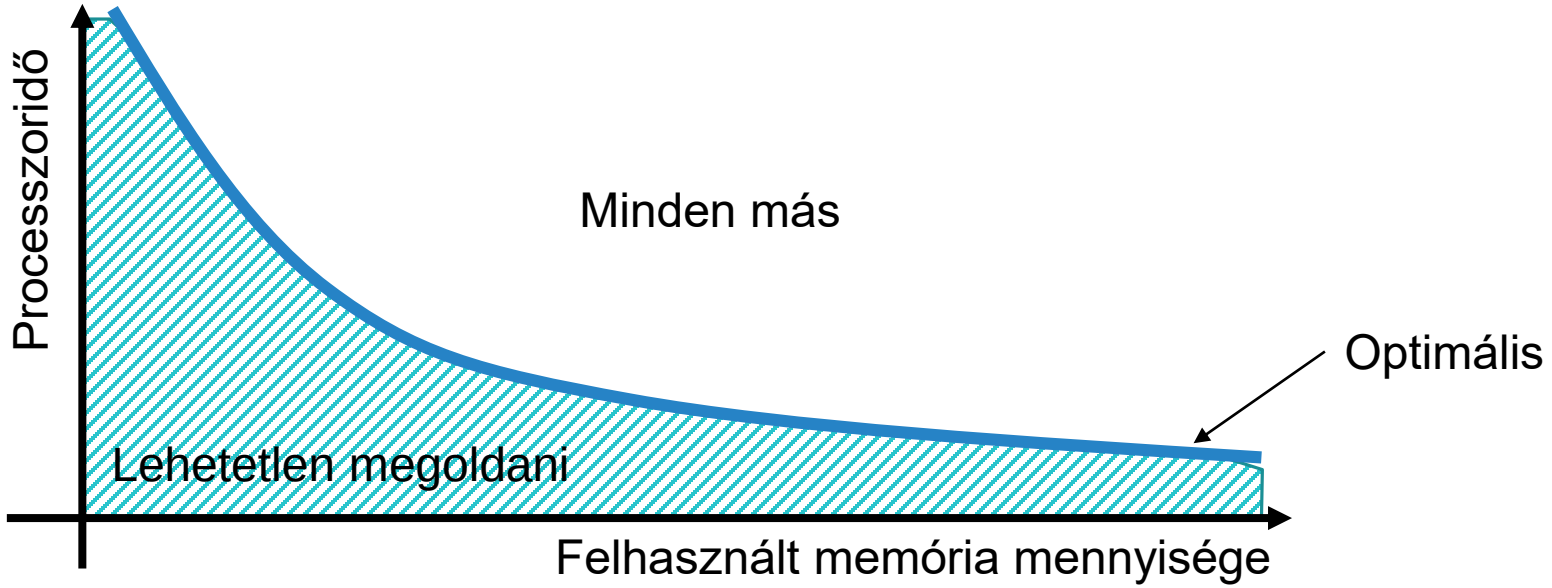
Hordozhatóság

- A program hordozhatóságának mértéke mutatja meg, mennyire könnyű a programot átalakítani
  - Más géphez
  - Más konfigurációhoz
  - Más operációs rendszerhez
  - Általában más környezethez

Minőségi szoftver

Hatékonyság

- A program hatékonysága a futási idővel és a felhasznált memória méretével, illetve *további erőforrások* használatával arányos.
  - Egyszerűsítve: egy program hatékony, ha gyorsan lefut és kevés memóriát használ.



Minőségi szoftver

Barátságosság

- A program emberközelisége, barátságossága a felhasználó számára rendkívül fontos
  - Megköveteli, hogy legyen
    - az input logikus és egyszerű
    - az eredmények formája áttekinthető

Tesztelhetőség

- A tesztelhetőség, áttekinthetőség a program karbantartói, fejlesztői számára fontos.

Modularitás

Programozási módszertanok

Moduláris tervezés

- Programjainkat egymástól minél inkább független, de jól definiált kapcsolatban álló programegységek segítségével tervezzük.

Paradigmák

- Procedurális
- Objektumorientált
- Funkcionális
- Logikai
- Szimbolikus
- …

Moduláris dekompozíció

Jelentése

- A feladat több egyszerűbb részfeladatra bontása
- A részfeladatok megoldása egymástól függetlenül elvégezhető
- Ennek segítségével csökkentjük a feladat bonyolultságát.

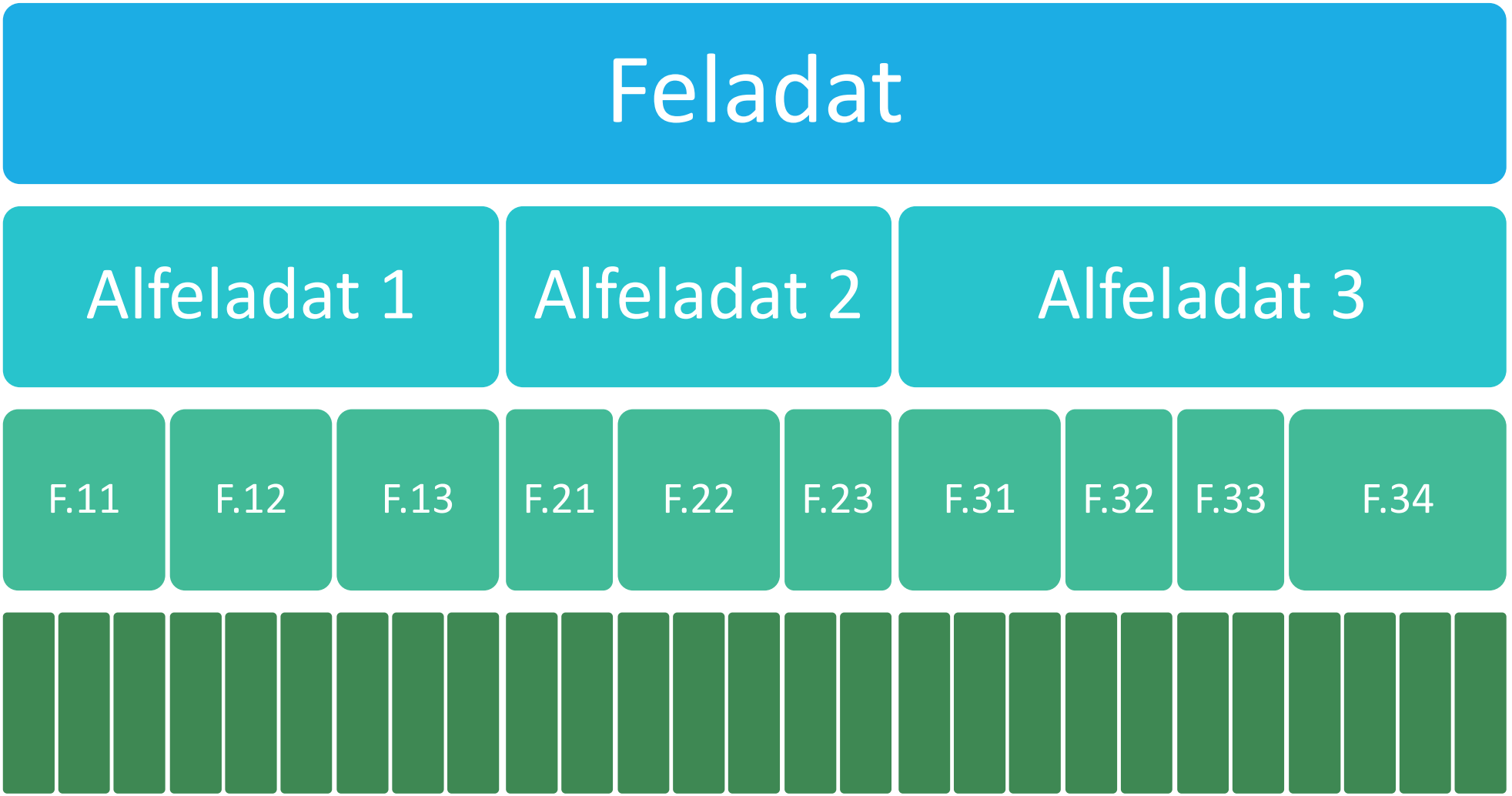
Általában a módszert ismételten alkalmazzuk

- Azaz az alrendszereket magukat is dekomponáljuk.
- Ez lehetővé teszi, hogy a feladat megoldásán egyszerre többen dolgozzanak

A módszer általi felbontás egy fával megadható.

- A fa csomópontjai az egyes dekompozíciós lépések

Moduláris dekompozíció

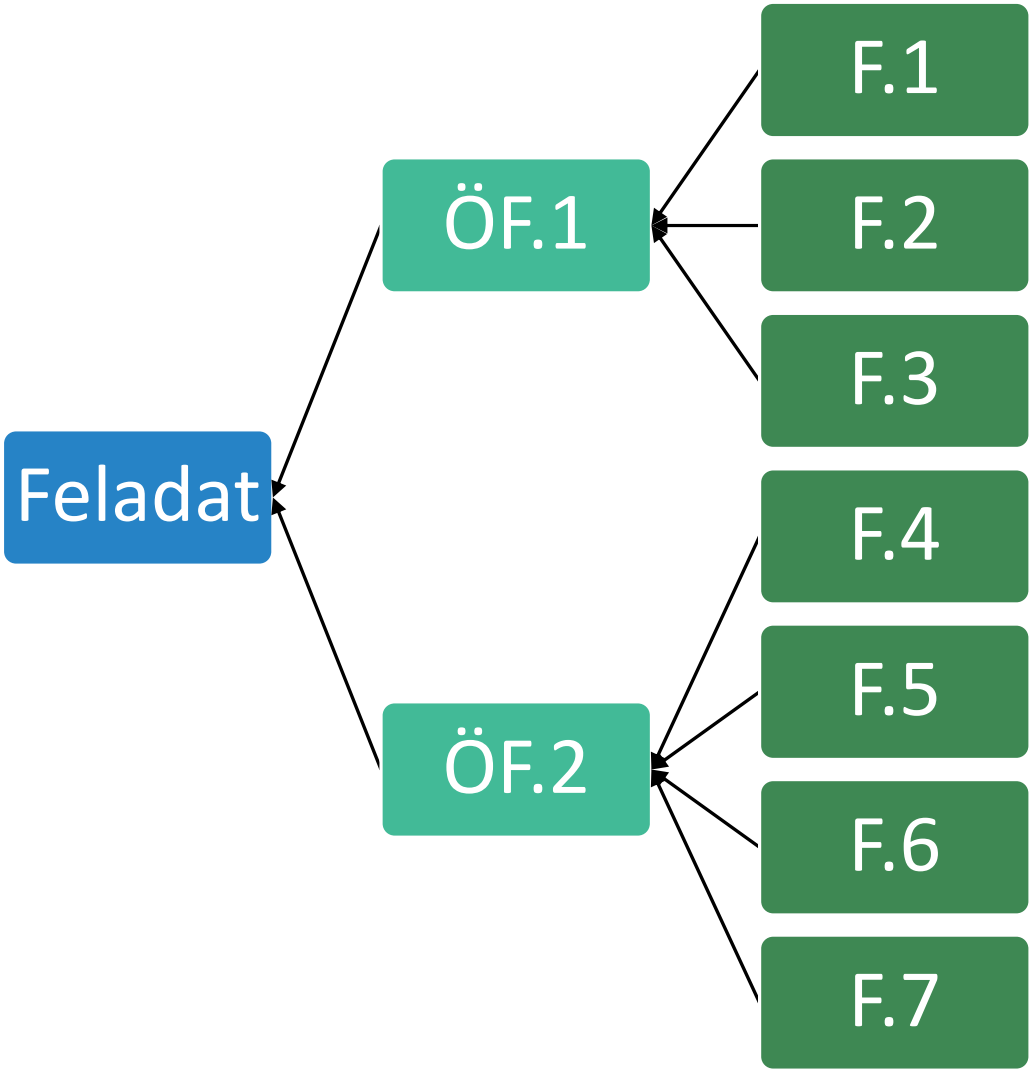


Moduláris kompozíció

Olyan szoftver elemek létrehozását támogatja, amelyek szabadon kombinálhatók egymással.

- A programok lehetőleg a már meglévő programegységekből, mint építőkövekből épülnek fel.

Moduláris kompozíció



Modularitás

Érthetőség

- A modulok önállóan is egy-egy értelmes egységet alkotnak
  - Megértésükhez minél kevesebb „szomszédos” modulra van szükség

Folytonosság

- A specifikáció „kis” változtatása
  → a programban is csak „kis” változtatásra van szükség

Védelem

- Célunk a program egészének a védelme az abnormális helyzetek hatásaitól.
  - Egy hiba hatása egy – esetleg néhány – modulra korlátozódjon

A modularitás alapelvei

A modulokat nyelvi egységek támogassák

- A modulok illeszkedjenek a használt programozási nyelv szintaktikai egységeihez.

Kevés kapcsolat legyen

- Minden modul minél kevesebb másik modullal kommunikáljon!

Gyenge legyen a kapcsolat

- A modulok olyan kevés információt cseréljenek, amennyi csak lehetséges!

Explicit interface kell

- Ha két modul kommunikál egymással, akkor annak ki kell derülnie legalább az egyikük szövegéből.

A modularitás alapelvei

Információ elrejtés

- Minden információ egy modulról rejtett kell legyen, kivéve, amit explicit módon nyilvánosnak deklaráltunk.

Nyitott és zárt modulok

- Zárt modul: más modulok számára egy jól definiált felületen keresztül elérhető, a többi modul változatlan formában felhasználhatja.
- Nyitott modul: kiterjeszthető
  - az általa nyújtott szolgáltatások bővíthetők vagy
  - hozzávehetünk további mezőket a benne levő adatszerkezetekhez, s ennek megfelelően módosíthatjuk eddigi szolgáltatásait.

Újrafelhasználhatóság

A típusok változatossága
- A modulok többféle típusra is működjenek.

Adatstruktúrák és algoritmusok változatossága

Egy típus – egy modul
- Egy típus műveletei kerüljenek egy modulba.

Reprezentáció-függetlenség

Programkönyvtárak

Újrafelhasználható programkönyvtárak

Tervezési igények

- Az alap-követelmények ugyanazok, mint más szoftver tervezésénél
  - Helyesség
  - Megbízhatóság
  - Karbantarthatóság
  - Újrafelhasználhatóság
  - Kompatibilitás
  - Hordozhatóság
  - Hatékonyság
  - stb.
- A programkönyvtár felé az igények erőteljesebbek!

Újrafelhasználható programkönyvtárak

Osztályokkal kapcsolatos további elvárások:

- Könnyű és intuitív használat
  - Könnyen megérthető legyen, néhány használat után jól megjegyezhető
- Azonnali, széleskörű felhasználhatóság
  - Szolgáltatásokat nyújt
- A lehető leghatékonyabb megvalósítás – state-of-the-art
- A tesztelés körültekintősége és az osztály megbízhatósága
- Magas szintű dokumentáció
  - Pontos, jól szervezett, hogy a felhasználó könnyen eligazodjon
- Platform-függetlenség
- Sokszor nyelv-függetlenség
- Meg kell hagyni a későbbi fejlesztés lehetőségét

Újrafelhasználható programkönyvtárak

Néhány következmény

- Konzisztencia
  - a könyvtár minden komponense egy átfogó, koherens tervezésnek felel meg
  - számos szisztematikus, explicit és egységes konvenciót követ
- Jó minőségű, homogén komponensek kellenek
- OOP megközelítés az adatabsztrakció támogatása miatt különösen is alkalmas a könyvtárak készítésére

Ezek egyidejű teljesítése nehéz

- Lehetséges, hogy a legáltalánosabb algoritmus nem a leghatékonyabb sok speciális, de gyakran előforduló esetben.

Újrafelhasználható programkönyvtárak

A könnyű felhasználhatóságnak különböző aspektusai vannak

- Az egyik ezek közül az osztályok elkülöníthetősége
  - Ha nagyon sok hasonló osztály található egy könyvtárban, akkor a felhasználó számára igen nehéz a pontosan megfelelő kiválasztása.
- Általában előnyösebb a kevés számú, majdnem teljesen ortogonális osztály definiálása
  - Minden esetben egy jól meghatározott szerep betöltésére
    - Az optimális megvalósítást követve

Fejlesztő tulajdonságai

A jó könyvtárfejlesztő tulajdonságai

- Van absztrakciós készsége
  - Képes arra, hogy az egyedi jelenségekből általánosítson
- Van érzéke a részletekhez
  - Egy könyvtárban minden kis tulajdonság számít.
  - Itt nincs olyan, hogy 'elég jó’
    - Teljes kell legyen a könyvtár
- Rend-mániákus
  - Képes az osztályozásra, hogy mindennek megtalálja a helyét.

Fejlesztő tulajdonságai

Van irodalmi érzéke

- A leírásainak, a specifikációs részeket magyarázó megjegyzéseknek legyen stílusa és eleganciája

Elsőrendű programozó és tervező kell legyen

- Tudja átlátni a könyvtár későbbi használójának igényeit

„Ego-less” legyen

- Nem szabad, hogy egyéni stílus érvényesüljön
  - A cél a konzisztens stílus használatának lehetősége
- A kreativitása a könyvtár által nyújtott szolgáltatásokban jöjjön elő!

Programkönyvtárak jellemzése

Egy programkönyvtár osztályok gyűjteménye.

- Egy osztályt az általa nyújtott szolgáltatások jellemeznek.

Ezen szolgáltatásokat a következő két módon lehet csoportosítani:

- I.
  - Számított (rutin)
    - Függvény
    - Parancs
  - Tárolt
    - Attribútum

Programkönyvtárak jellemzése

Egy programkönyvtár osztályok gyűjteménye.

- Egy osztályt az általa nyújtott szolgáltatások jellemeznek.

Ezen szolgáltatásokat a következő két módon lehet csoportosítani:

- II.
  - Értéket visszaadó
    - Számított – függvény
    - Tárolt – attribútum
  - Parancs

Programkönyvtárak jellemzése

Osztályok csoportosítási lehetősége

- csomagok (Java)
- clusterek (Eiffel)

használata.

A könyvtár és a csomag fogalmak kapcsolata lehet

- 1-sok
- sok-1

A csomagok gyakran a fizikai fájlrendszer struktúráját követik.

Osztályhierarchia

OO esetben egy programkönyvtár elkészítése egy osztályhierarchia kialakítását is jelenti.

- A használhatóság szempontjából nem lényegtelen ezen hierarchia bonyolultsága vagy egyszerűsége.

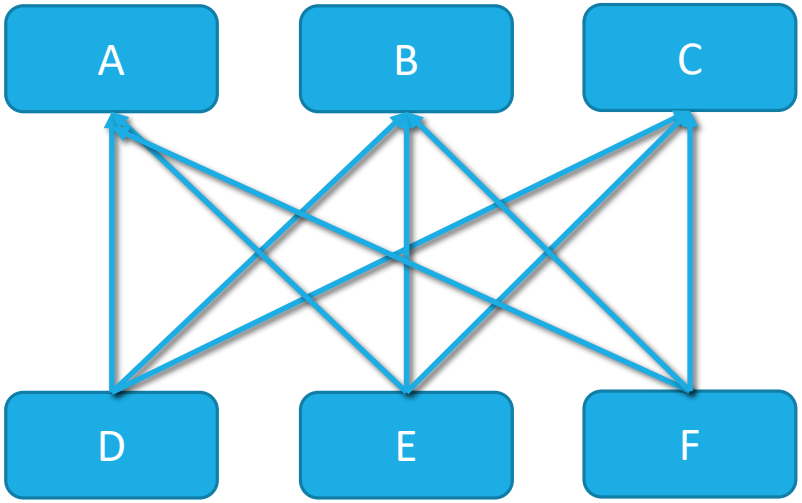
Az objektumorientált szemléletnek megfelelően

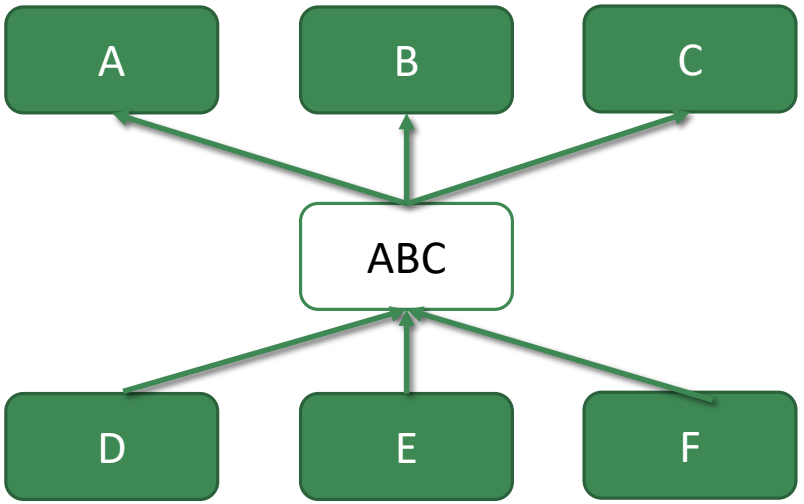
- igyekezzünk minél általánosabb osztályokat bevezetni (generalizáció)
- az öröklődést felhasználva hozzuk létre a szükséges osztályokat (specializálódás).

Ezáltal az osztályhierarchia karbantartása is leegyszerűsödik.

Osztályhierarchia

Új osztály bevezetésének másik oka az osztályhierarchia egyszerűsítése is lehet:





Biztonság

Egy sokak által használt programkönyvtár esetén fontos szempont a megbízhatóság és a biztonság.

- elő- és utófeltételek megadása
- invariánsok adásával

„Szerződés” létrejötte

- az előfeltétel a felhasználó kötelezettsége és a programozó biztosítéka
- az utófeltétel a programozó kötelezettsége és a használó biztosítéka
- Nem kell „defenzíven” programozni
  - minden lehetséges hibalehetőségre és bemenetre felkészülve
  - a kód hatékonyabb és egyszerűbb lesz

Biztonság

Két megközelítési mód

- Toleráns
  - A programkönyvtári rutinoknak nincs (vagy csak gyenge) előfeltételük
  - Minden lehetséges bemenetre valamilyen módon reagálnak
- Követelőző
  - Minden rutin szigorú előfeltételekkel rendelkezik
  - Ezek biztosítása a felhasználó felelőssége

Az ajánlott megközelítés:

- Csak az absztrakt művelet logikai helyes elvégzéséhez szükséges előfeltételeket ellenőrizzük.
- Csak azt ellenőrizzük, ami, ha nem teljesülne, súlyosan befolyásolná a hatékonyságot.

Biztonság

Az így létrejött szerződés bármilyen megsértése programhibát jelent

- Az előfeltétel megsérülése felhasználó oldali hiba
- Az utófeltétel vagy az invariáns sérülése programkönyvtár hiba

A programkönyvtár fejlesztése után az előfeltétel teljesülés vizsgálatát kell elvégezni

- A terjesztett változatban a hatékonyság érdekében.

Dokumentáció

Egy programkönyvtár több fejlesztő munkája.

A felhasználhatóság szempontjából ezért igen fontos a jó dokumentáció.

Lehetséges megközelítési módok:

- „Forrásszöveg maga a dokumentáció, mi kell még?”
- Különálló dokumentáció írása

Példák

```
/**
 * Always returns true.
 */
public boolean isAvailable() {
    return false;
}
```

```
for (j=0; j<arra_len; j+ =8) {
  total += array[j+0];
  total += array[j+1];
  total += array[j+2]; /* comment
  total += array[j+3];  * out
  total += array[j+4];  * some
  total += array[j+5];  * lines
  total += array[j+6];  */
  total += array[j+7];
}
```

```
#define TRUE FALSE
```

```
a=a++
```

```
*++b ? (*++b + *(b-1)) 0
```

- Nem definiált viselkedés, a precedencia miatt

Híres példa – $1/\sqrt{(x)}$

```
float Q_rsqrt( float number )
{
    long i;
    float x2, y;
    const float threehalfs = 1.5F;

    x2 = number * 0.5F;
    y = number;
    i = * ( long * ) &y; // evil floating point bit level hacking
    i = 0x5f3759df - ( i >> 1 ); // WTF?
    y = * ( float * ) &i;
    y = y * ( threehalfs - ( x2 * y * y ) );
                        // 1st iteration
// y = y * ( threehalfs - ( x2 * y * y ) );
                        // 2nd iteration, this can be removed
    return y;
}
```

Dokumentáció

A forrásszöveg nem elég absztrakt

- A felhasználó számára fontos információkon kívül az alacsonyszintű implementációt is tartalmazza
- A használata túl sok információ átfésülését jelentené
- Arra sarkallná a programozókat, hogy nem publikusnak szánt implementációs lehetőségeket is kihasználjanak
- Előnye, hogy mindig aktuális.

Különálló dokumentáció esetén nem biztosított a szoftver és a dokumentáció konzisztenciája.

- Előnye, hogy csak a felhasználó számára fontos dolgokat tartalmazza.

Dokumentáció

Belső dokumentáció elve: a szoftver dokumentációja a programszövegbe legyen beágyazva.

Ennek előnyei:

- Forrásszöveg és dokumentáció mindig konzisztens marad.
  - Előbbi példa …
- Dokumentáció kinyerése automatizálható
  - javadoc

Eiffel flat-short forma

- az osztálynak egy olyan verziója, ami tartalmazza az öröklött és az itt bevezetett jellemzőket
  - figyelembe véve minden átnevezést és átdefiniálást
  - felépítve a teljes elő- és utófeltételt és a típusinvariánst

Karbantartás

Egy programkönyvtár állandó fejlesztés alatt áll,

- Lehetőséget kell biztosítani újabb verziók problémamentes beépítésére.

A következő változtatások nem jelentenek problémát:

- Osztály bővítése új szolgáltatással.
- Egy szolgáltatás implementációjának lecserélése
- Előfeltétel gyengítése, illetve utófeltétel szigorítása

Karbantartás

Szükség lehet egy szolgáltatás új verziójának bevezetésére, ha

- A szolgáltatás neve megváltozik
  - Például egységes névkonvenció kialakítása miatt
- Jobb szignatúra vagy specifikáció megadása
- Rutin elavulttá nyilvánítása.
  - Ez történhet nyelvi kulcsszavak használatával
    - Eiffel: obsolete
    - Java: deprecated
  - Vagy a régi változat átírásával
    - Az egyből az új változatot hívja meg (hívásátirányítás)
    - Esetleges figyelmeztető üzenet kiírása mellett

Osztályok mérete

Egy osztály méretét a következő módokon lehet meghatározni:

- Sima (teljes) méret = szolgáltatások száma (öröklött + itt bevezetett).
- Közvetlen méret = közvetlen (nem absztrakt, illetve újradefiniált) szolgáltatások száma.
- Növekményes méret = közvetlen és újradefiniált szolgáltatások száma.

Másik szempont lehet, hogy tartalmazza-e a nem exportált jellemzőket:

- Külső méret
- Belső méret

Osztályok mérete

A teljes méretre nyilván nincs felső határ

- A közvetlen, és a növekményes méret az, ami szerepel az osztály szövegében.
- De egy adott méreten felül az osztály kezelhetetlen.

Két megközelítési lehetőség

- Minimalista nézőpont
  - egy programkönyvtár osztálya a megvalósított típusnak csak a legalapvetőbb (atomi) szolgáltatásait tartalmazza,
  - redundáns (azaz ami atomi szolgáltatások használatával is megvalósítható) metódusokat pedig nem.
  - Ez persze a programkönyvtár írójának kedvez, hiszen kevesebb munkát és egyszerűbb karbantartást jelent.

Osztályok mérete

- Bevásárlólista nézőpont: minden olyan szolgáltatást vegyünk bele az osztályunkba, ami az alábbi szempontoknak megfelel:
  - A szolgáltatás beleillik az absztrakt adattípus megvalósítási sémájába.
  - Nem rontja el az osztály helyességét, azaz az osztályinvariánst.
  - Hasznos funkcionalitást valósít meg.
  - Nem duplikál valamely már meglévő szolgáltatást.
  - Megfelel a könnyen kezelhetőség kritériumainak (lásd később).

Osztályok mérete

A gyakorlatban általában

- a programozási nyelvtől elvárt szempont a minimalista nézőpont
  - azaz hogy műveletet csak minél kevesebb módon lehessen hatékonyan elvégezni,
- míg a programkönyvtárak tervezésekor a bevásárlólista nézőpont érvényesül
  - a felhasználás megkönnyítése érdekében

Szolgáltatások száma

Túl sok szolgáltatás a programkönyvtár használatának megtanulhatóságát is kedvezőtlenül befolyásolja

Ezért

- Minden szolgáltatás egy jól meghatározott szerződéssel definiált
  - csak a specifikációjuk, nem pedig az implementációjuk alapján kerülnek felhasználásra.
- Az osztálydokumentáció egységes és könnyen áttekinthető formátumú legyen
  - feltüntetve minden szolgáltatás szerződését és elrejtve az implementációs részleteket
- A szolgáltatások nevei szigorú elnevezési konvenciót kövessenek.
- A szolgáltatásokat csoportosítsuk pontosan definiált kategóriák szerint, ezen kategóriák minden osztálynál egyezzenek meg
  - sorrendjük a dokumentációban legyen azonos
  - kategórián belül a szolgáltatások névsorrendben következzenek

Példaadatok

Általában kijelenthető, hogy 80 szolgáltatásnál többet tartalmazó osztály esetén már meggondolandó, hogy nem lehetne-e az osztályhierarchiát tovább bontani.

Szolgáltatások mérete

Fontos, hogy a használó pontosan ismerje egy szolgáltatás paramétereinek számát, típusát és sorrendjét.

- Ezért a felhasználás szempontjából egy szolgáltatás méretét
  - Tulajdonképpen a megadandó paraméterek száma
- Ez a méret erősen függ a programkönyvtár feladatától is,
  - Egy grafikus felületet megvalósító, vagy statisztikai számításokat végző osztály metódusai rengeteg argumentumot igényelhetnek.

Szolgáltatások mérete

Nagyszámú argumentum esetén érdemes azokat két csoportra osztani:

- Paraméter: olyan argumentum, amelynek értékével valamilyen műveletet kell elvégezni.
- Opció: olyan argumentum, amely akár el is hagyható, ugyanis rendelhető hozzá egy alapértelmezett érték.
  - Csak a paraméterek feldolgozását befolyásolja.

Egy szolgáltatás fejlesztése során a paraméterlista lehetőleg ne változzon

- Új opciókat nyugodtan fel lehet venni.

Szolgáltatások mérete

A szolgáltatás méretének csökkentése érdekében csak paramétereket vegyünk fel az argumentumlistába.

Az opciókat pedig külön attribútummal reprezentáljuk

- Adjunk hozzá új beállító/lekérdező szolgáltatásokat.
  - Opció beállítási technika.
- Növeli az osztály méretét
  - Esetleges új, globális opció-attribútum

Példaadatok

Általában kijelenthető, hogy az a jó, ha a szolgáltatások átlagos paraméterszáma 0.5

- 5-nél több paraméter esetén meggondolandó a továbbontás

A szolgáltatások lekérdezés – parancs javasolt aránya

- 60 - 40%

Elnevezés

Egy programkönyvtár használatát nagyban elősegíti az egységes elnevezési konvenció.

- Ez konzisztens névhasználatot jelent
- Kialakítása a megvalósított adattípusok közös tulajdonságain alapul

Például

- egy adott elem elérésére (get)
- egy elem megváltoztatására (put)
- egy elem törlésére (remove)

Azonos neveket használunk különböző osztályokon belül.

- Mégsem vezet félreértéshez
  - a típusok különbözősége miatt (általában) a szignatúrák nem egyeznek meg

Elnevezés

Standard elnevezések:

- `capacity` - adott struktúra kapacitását adja vissza.
- `count` - adott struktúra elemszámát adja vissza.
- `empty` - megadja, hogy az adott adatstruktúra üres-e.
- `full` - megadja, hogy az adott adatstruktúra tele van-e.
- `get` - struktúra adott elemét adja vissza.
- `has` - lekérdezi, hogy adott elem a struktúra eleme-e.
- `put` - struktúra adott elemét állítja be.
- `remove` - struktúra adott elemét kitörli.

Elnevezés

Irányelvek:

- A név legyen rövid (rendszerint egy szó), de beszédes.
- Szolgáltatás neve ne tartalmazzon utalást a szolgáltatást tartalmazó osztályra
  - kivéve többszörös öröklődés esetén, ha átnevezés kell
- Osztály neve mindig főnév(i szerkezet) legyen.
- Parancsok (eljárások) nevei legyenek (felszólító módú) igék
  - Esetleges főnévi vagy értelmező jelzői kiegészítővel
- Nem logikai lekérdezések nevei legyenek főnevek vagy főnévi szerkezetek.

Elnevezés

- Logikai lekérdezés neve legyen olyan melléknév, amely igen/nem választ sugall
  - Vagy használjuk az is_ (-e) szerkezetet
- A szolgáltatás nevét a célobjektum szemszögéből lehessen értelmezni
  - A szolgáltatás hívásakor mindig létezik egy objektum (a célobjektum) aminek a szolgáltatásáról szó van.
  - has
- Összetartozó művelet és lekérdezés párok elnevezéseinek szótöve legyen azonos
  - extendible - extend

Osztályok fajtái

Konkrét típusok

- Minden programkönyvtár hierarchiában találunk olyan típusokat, melyek alapvető adatstruktúrákat reprezentálnak
- lista, vektor, string, dátum, komplex szám, …

Osztályok fajtái

Konkrét típusok – tulajdonságok

- Szoros illeszkedés egy konkrét fogalomhoz és a megvalósítás módjához.
- Érthetőség, önálló használhatóság.
- Erősen implementációfüggők
  - Ezért hatékony a megvalósításuk
  - Bármilyen módosításukkor minden felhasználói kódot újra kell fordítani.
- Minimális mértékben függenek más osztályoktól.
- Különállóan (izoláltan) is fordíthatóak és használhatóa
- Nem, illetve ritkán lehet tőlük örökölni.

Osztályok fajtái

Absztrakt típusok

- Szerepük rendszerint egy adott specifikáció (akár interfész is lehetne) bevezetése. Pl. halmaz
- Az absztrakt típusokat konkrét típusok segítségével lehet implementálni.

Absztrakt típusok – tulajdonságok

- Azonos elérési módhoz többfajta implementáció is létezhet.
- Hatékony tárkihasználás és elfogadható futási idő a virtuális metódusok segítségével.
- Az implementáció minimális mértékben függ más osztályoktól.
- Különállóan is lefordíthatóak – érthetőek önmagukban.

Osztályok fajtái

Csomópont típusok

- Az öröklési hierarchia belső pontjait alkotják.

Jellemzőik

- Az ősosztályok szolgáltatásait implementálja,
  - Interfészét kibővíti virtuális metódusokkal is, melyeket maga is implementál.
- Függ az őseitől.
- Lehet belőle örökölni.
- Példányosítható.

Osztályok fajtái

Felületosztályok

- Egy felület igazítása úgy, hogy jobban illeszkedjen a felhasználó elvárásaihoz.
  - Például vektor ne 0-tól legyen indexelve.
- Ellenőrzött vagy korlátozott felületek biztosítása.

Osztályok fajtái

„Kövér interfészek”

- Ez egy olyan típus, amely nem függ más osztályoktól, rengeteg szolgáltatást vezet be
  - ezek közül csak a legalapvetőbbekhez ad implementációt, míg a speciálisabb szolgáltatásokat virtuálisként deklarálja
  - Lehet példányosítani, ad hozzá egy üres implementációt, ami rendszerint egy hibaüzenet kiírását jelenti
    - Jelzi, hogy az adott szolgáltatás nem implementált.
    - Példa: általános tároló (container) osztály.

Osztályok fajtái

Keret osztályok (Framework)

- Ezen absztrakt osztályok tulajdonképp egy mini alkalmazást valósítanak meg.
- Maga a programlogika van csak az osztályon belül implementálva
  - minden egyéb paraméter-bekérő és alapműveletet elvégző metódus virtuális
- Példa: szűrő osztály, amely egy bemeneti adatfolyam minden elemén végiglépkedve adott feltétel teljesülésekor bizonyos műveleteket elvégez.
- A bemeneti adatfolyamon történő végigléptetés, adott tulajdonság fennállásának ellenőriztetése,
  - A kívánt művelet meghívása és esetleges hibakezelés ezen osztályban van implementálva.

Osztályok fajtái

Leíró osztályok

- Felmerülhet az igény adott interfészen keresztül változó méretű osztályok objektumainak kezelésére is.
  - Erre ad megoldást a leíró osztályok használata.
- Ekkor ugyanis az implementáció két részre bomlik
  - az aktuális reprezentációra
  - a reprezentáció elérését biztosító leíró (handle) objektumra
  - Ez tulajdonképp a mutató típus objektumorientált megvalósításának tekinthető.
- Példa: fájlleíró osztályok

Memóriakezelés

A programkönyvtár tervezés egyik kulcsfontosságú kérdése.

Két alapvető probléma:

- Memóriafoglalás új objektumok létrehozásakor.
- Memória felszabadítása.

Míg az első az operációs rendszer feladata, addig a második problémára két megoldási irányzat is született:

- Automatikus szemétgyűjtés (pl. Java).
  - Ez rendszerint referenciák bevezetését és a mutató típus megszüntetését jelenti.
- Objektumok kézzel történő felszabadítása.
  - Hatékonyabb, de sokkal veszélyesebb.

Algoritmus, grammatika, fordítóprogramok

JÖVŐ HÉTEN