Programozási nyelvek és módszerek

10. ELŐADÁS – PROGRAMKÖNYVTÁRAK

Újrafelhasználható programkönyvtárak

Tervezési igények

- Az alap-követelmények ugyanazok, mint más szoftver tervezésénél
  - helyesség
  - hatékonyság
  - megbízhatóság
  - kiterjeszthetőség
  - stb.,
- De a könyvtár felé az igények erőteljesebbek!

Újrafelhasználható programkönyvtárak

Az újrafelhasználható könyvtárakban található osztályokkal kapcsolatban az alábbi tulajdonságok várhatóak el:

- Könnyű és intuitív használat
  - könnyen megérthető legyen, néhány használat után jól megjegyezhető
- Azonnali, széleskörű felhasználhatóság szoftverrendszerek széles körében
  - szolgáltatásokat nyújt
- A kurrens technológiához mért lehető leghatékonyabb megvalósítás

Újrafelhasználható programkönyvtárak

- 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

Ezen követelmények néhány következménye:

- konzisztencia - a könyvtár minden komponense egy átfogó, koherens tervezésnek kell megfeleljen, és számos szisztematikus, explicit és egységes konvenciót kell kövessen.
- jó minőségű, homogén komponensek kellenek
- az objektum-orientált megközelítés az adatabsztrakció támogatása miatt különösen is alkalmas a könyvtárak készítésére

Újrafelhasználható programkönyvtárak

A fenti kritériumok egy részének egyidejű teljesítése igen nehéz.

- könnyen 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.
  - Amennyiben nagyon sok hasonló osztály található egy könyvtárban, akkor a felhasználó számára igen nehéz a pontosan megfelelő kiválasztása.
- Általában elmondható, hogy előnyösebb kevés számú, majdnem teljesen ortogonális osztály definiálása
  - minden esetben egy jól meghatározott szerep betöltésére, s az optimális megvalósítást követve.

Tulajdonságok

A jó könyvtárfejlesztő tulajdonságai

- van absztrakciós készsége, képes arra, hogy az egyedi jelenségekből általánosítson
- van érzéke ugyanakkor a részletekhez
  - egy könyvtárban ui. 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.

Tulajdonságok

Van irodalmi érzéke

- A leírásainak, a specifikációs részeket magyarázó megjegyzés-sorainak 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 másfajta csoportosítási lehetősége a csomagok (Java) vagy clusterek (Eiffel) használata.

A könyvtár és a csomag fogalmak 1-sok, illetve sok-1 kapcsolatban állhatnak egymással.

- Rendszerint a csomagok egy az egyben képződnek le fizikai tárolóterületre, például könyvtárszerkezetre.

Osztályhierarchia

OO esetben egy programkönyvtár elkészítése egy osztályhierarchia kialakítását is jelenti.

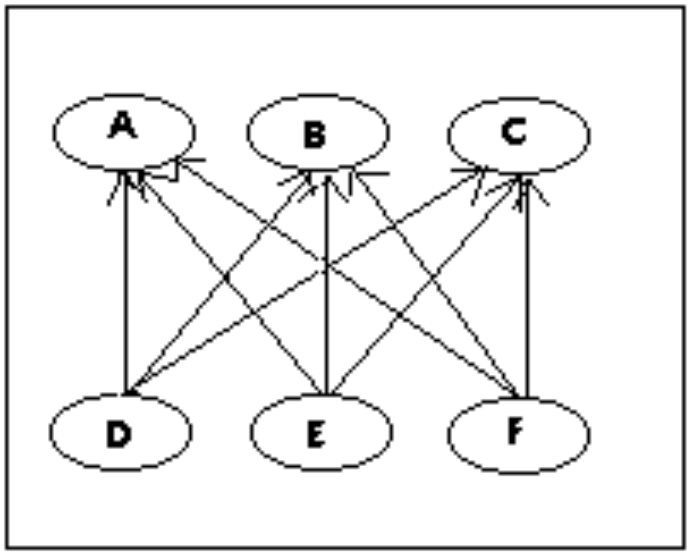
- A használhatóság szempontjából nem lényegtelen ezen hierarchia bonyolultsága vagy egyszerűsége.

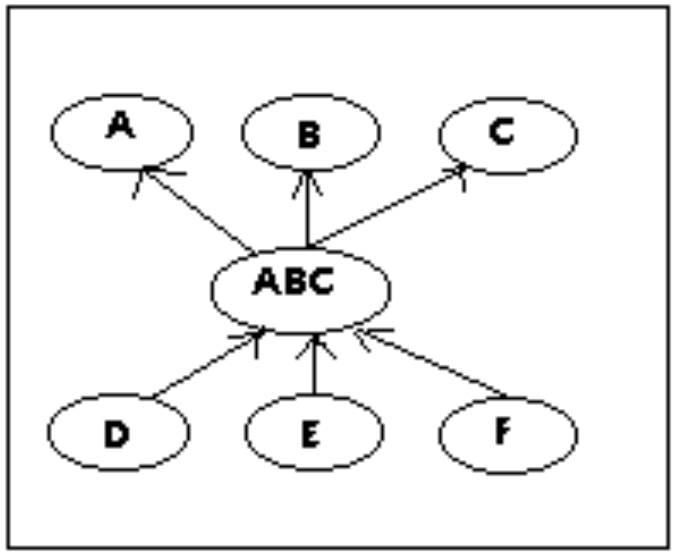
Az objektumorientált szemléletnek megfelelően igyekezzünk minél általánosabb osztályokat bevezetni (generalizáció) és az öröklődést felhasználva hozzuk létre a szükséges osztályokat (specializálódás).

- 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.

Ennek biztosítása elő- és utófeltételek, valamint invariánsok adásával valósítható meg.

Ez – a programkönyvtár írói és használói között – egy szerződés létrejöttét jelenti, ahol az előfeltétel a felhasználó kötelezettsége és a programozó biztosítéka, míg az utófeltétel a programozó kötelezettsége és a használó biztosítéka.

Osztályhierarchia

Így a programozónak nem kell "defenzíven" programoznia, (minden lehetséges hibalehetőségre és bemenetre felkészülve) és a kód is hatékonyabb és egyszerűbb lesz.

A lehetséges kétfajta megközelítési mód

- Toleráns megközelítés: a programkönyvtári rutinoknak nincs (vagy csak gyenge) előfeltételük és minden lehetséges bemenetre valamilyen módon reagálnak.
- Követelődző megközelítés: 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.

Az így létrejött szerződés bármilyen megsértése programhibát jelez.

- Az előfeltétel megsérülése felhasználó oldali hibát, míg az utófeltétel vagy az invariáns sérülése programkönyvtár hibát jelez.
- Ezért a programkönyvtár fejlesztése után a terjesztés előtt a hatékonyság érdekében elég csak az előfeltétel teljesülés vizsgálatát bekapcsolni.

A lényeg

Explicit megszorítás kell, hogy a felhasználó tudja, mire számíthat!

Dokumentáció

Egy programkönyvtár több fejlesztő munkája.

A felhasználhatóság szempontjából ezért igen fontos a jó dokumentáció.

Lehetséges megközelítési módok:

- Forrásszöveg?
- Különálló dokumentáció?

Dokumentáció

A forrásszöveg nem elég absztrakt.

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

Dokumentáció

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.
  - ?!
- 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

Mivel egy programkönyvtár állandó fejlesztés alatt áll, ezért lehetőséget kell biztosítani újabb verziók problémamentes beépítésére.

A felhasználás szempontjából a következő változtatások nem jelentenek problémát:

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

Karbantartás

A következő esetekben lehet szükség egy szolgáltatás új verziójának bevezetésére

- Szolgáltatás nevének megváltoztatása
  - 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 oly módon, hogy az egyből az új változatot hívja meg (hívásátirányítás), esetleges figyelmeztető üzenet kiírása kíséretében.

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.

Osztályok mérete

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, ill. a növekményes méret az, ami szerepel az osztály szövegében.

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 féleképpen lehessen hatékonyan elvégezni,
- míg a programkönyvtárak tervezésekor a felhasználás megkönnyítése érdekében inkább a bevásárlólista nézőpont érvényesül.

Szolgáltatások száma

Túl sok szolgáltatás a programkönyvtár használatának megtanulhatóságát is kedvezőtlenül befolyásolja, ennek kezelhetősége érdekében a következő rendezőelvek betartása ajánlott:

- 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.

Szolgáltatások száma

- 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

Mivel a programkönyvtárak használata a megvalósított szolgáltatások meghívásából áll, ezért igen 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 jellemzi.

Ez a méret erősen függ a programkönyvtár feladatától is, hiszen egy grafikus felületet megvalósító, vagy statisztikai számításokat végző osztály metódusai rengeteg argumentumot igényelhetnek.

Szolgáltatások mérete

Nagyszámú argumentum esetén érdemes azokat a következő 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, míg új opciókat nyugodtan fel lehet venni.

Szolgáltatások mérete

Ezekből következik, hogy a szolgáltatás méretének csökkentése érdekében csak paramétereket vegyünk fel az argumentumlistába.

Az opciókat pedig külön attribútummal reprezentáljuk és adjunk hozzá új beállító/lekérdező szolgáltatásokat.

- Ez az ún. opció beállítási technika.
- Ez persze felveti az új opció-attribútum globalitásának kérdését, sőt növeli az osztály méretét is.

Példaadatok

Általában kijelenthető, hogy az a jó, ha a szolgáltatások átlagos paraméterszáma 0.5, és 5-nél több paraméter esetén meggondolandó, hogy nem lehetne-e a szolgáltatást tovább bontani.

A szolgáltatások lekérdezés – parancs javasolt aránya

- 60 - 40%

Elnevezés

Egy programkönyvtár használatát nagyban elősegíti az egységes elnevezési konvenció.

Ez konzisztens névhasználatot jelent, kialakítása padig azon alapul, hogy kategorizáljuk a megvalósított adattípusok közös tulajdonságait

Elnevezés

Például

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

Így azonos neveket használunk különböző osztályokon belül, de ez mégsem vezet félreértéshez, mert a típusok különbözősége miatt általában a szignatúrák sem egyeznek meg.

Elnevezés

Ilyen elnevezés-konvenció bevezetésekor szinte biztos, hogy használni fogjuk a következő standard elnevezéseket:

- `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, mivel egy 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

Az osztályok fajtái

- Konkrét típusok
- Absztrakt típusok
- Csomópont típusok
- Felület osztályok
- Kövér interfészek
- Alkalmazás keret
- Leíró osztályok

Az osztályok fajtái

Konkrét típusok

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

Az osztályok fajtái

A következő jellemzi:

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

Az osztályok fajtái

Absztrakt típusok

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

Tulajdonságaik:

- 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.

Csomópont típusok

Az öröklési hierarchia belső pontjait alkotják.
Jellemzőik:

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

Példa: ablak, négyszög...

Felületosztályok

Egy felület igazítása úgy, hogy jobban illeszkedjen a felhasználó elvárásaihoz.

- Pl. vektor ne 0-tól legyen indexelve.

Ellenőrzött vagy korlátozott felületek biztosítása.

„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
- hogy lehessen példányosítani, ad hozzá egy üres implementációt, ami rendszerint egy hibaüzenet kiírását jelenti
  - jelzi, hogy az adott szolgáltatás nem implementált.
  - Példa: általános tároló (container) osztály.

Alkalmazás keretek (frame)

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, illetve a kívánt művelet meghívása és esetleges hibakezelés ezen osztályban van implementálva.

Leíró osztályok

Míg egy absztrakt interfész és annak konkrét implementációja közötti kapcsolat a program futása során állandó, az implementáló osztály mérete is állandó, 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 és 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.

Fontos tanácsok

How To Write Unmaintainable Code

- https://thc.org/root/phun/unmaintain.html