Szakdolgozat

Nyíri Norbert

Debrecen

2006

Debreceni Egyetem
Informatikai Kar

Aspektus orientált programozás

Témavezető:
Espák Miklós
számítástechnikai munkatárs

Készítette:
Nyíri Norbert
programozó matematikus hallgató

Debrecen
2006

Tartalomjegyzék

Bevezetés

A szakdolgozat az aspektus orientált programozást mutatja be. Célja, hogy megismertesse az olvasóval az aspektus orientált paradigma alapvető fogalmait és gondolkodás módját. Ez egy teljesen új irányzat a programozás terén. Jelentősége abban rejlik, hogy az objektum orientált paradigma nyelveinek hiányosságait próbálja meg kitölteni, illetve hibáit javítani.

A dolgozat első részében magával az aspektus orientált programozással és annak alapvető fogalmaival fogunk foglalkozni. Ez elengedhetetlen ahhoz, hogy az egyes nyelvek működését és fogalmait a későbbiekben megérthessük.
A következő lépésben megfogalmazunk egy példát, melyen keresztül később, egy nyelv eszközeit részletesebben is tárgyalhatjuk, így téve könnyebbé az új eszközök megértését. A példa programot az aspektus orientált programozás szemléletmódjában megtervezzük, majd az egyes programozási nyelvek is bemutatásra kerülnek. Azért foglalkozunk először a program tervezésével, hogy szemléletessé váljon, hogy egy aspektus orientált program fejlesztése nem függ a nyelvtől, melyen később meg akarjuk valósítani. Az aspektus orientált programozás ugyanis nem csak új nyelvek létrejöttét jelenti, hanem egy új szemlélet és gondolkodás módot. Ennek megértését kívánja elősegíteni a program megtervezése. Ezzel könnyebbé válik a nyelvek megértése is.

A program megtervezése után három ismert aspektus orientált programozási nyelv kerül bemutatásra, majd végül a példa program megvalósítása következik. A kész program újdonságokat tartalmazó részének részletes bemutatásával.

Az aspektus orientált programozás

Általános fogalmak

Concern

A concern az aspektus orientált programozás egyik legfontosabb fogalma az Aspektus (aspect) mellet. Az angol concern szó magyar jelentése vonatkozás, törekvés, érdekeltség. Noha a szó jelentése már egyfajta definíciót ad a fogalomról, mégis fontos, hogy pontosabban megfogalmazzuk, a vonatkozás (concern) jelentését az aspektus orientált paradigmában. A szakirodalom azonban nem azonos módon definiálja a fogalmat. A legelterjedtebb és legmegfelelőbb definíciók:

Vonatkozás*:* Egy olyan dolog, ami egy szoftver rendszerben fontos számunkra, azaz egy olyan cél vagy tulajdonság halmaz melyet a rendszernek teljesítenie kell.

Vonatkozás: Egy olyan lényeges követelmény melyet a szoftver rendszerben kezelni kell, hogy a rendszer általános céljait elérjük

Vonatkozás*:* Egy olyan követelmény, amely a rendszer fejlesztését vagy üzemeltetését nagymértékben befolyásolja.

A különböző definíciók nagyon hasonlóak egymáshoz, s mindegyik megragadja a vonatkozás fogalmának lényegét. Mégpedig, hogy vonatkozásnak nevezzük a szoftver rendszerrel szemben támasztott azon megrendelői követelményeket, melyeket a rendszertervezőknek figyelembe kell venniük. Egyes nézetek szerint egy szoftver rendszer nem más, mint vonatkozások egy halmazának a megvalósítása. Épp ezért az olyan követelményeket melyek nem a rendszerre magára, hanem például a fejlesztésre vonatkoznak (pl.: költség és határidők) nem tekintjük vonatkozásnak.

Vonatkozások csoportosítása

A vonatkozásokat két nagy csoportba soroljuk alap vonatkozások (core concern) és kereszt vonatkozások (crosscutting concern).

Alap vonatkozás: A rendszer központi funkciói.

Kereszt vonatkozás: A rendszer azon funkciói melyek az alap vonatkozásokat vagy más kereszt vonatkozásokat metszenek. Azt mondjuk, hogy két vonatkozás keresztezi egymást, ha ezeknek a funkcióknak a műveletei metszik egymást.

Nem minden esetben teszünk különbséget alap és kereszt vonatkozások között.

Azon programozási nyelveket melyek nem tesznek különbséget lap és kereszt vonatkozások között szimmetrikus programozási nyelveknek hívjuk (pl.: Hyper/J [1]), míg azokat melyek különbséget tesznek köztük aszimmetrikus programozási nyelveknek nevezzük

Vonatkozások szétválasztása

A vonatkozások szétválása egy alapvető szoftvertervezési elv. Az a képesség, hogy a szoftver azon részeit felismerjük, összekapcsoljuk és manipuláljuk melyek egy meghatározott célra vagy követelményre vonatkoznak. Ahhoz, hogy megfelelő módon felismerjük és összekapcsoljuk a vonatkozásokat képző szoftver funkciókat, ismernünk kell a tulajdonságaikat is. A vonatkozásoknak egy erős belső összefüggése van, lehetőleg függetlenek más vonatkozásoktól, összekapcsolják az adatokat és az azokkal dolgozó metódusokat, valamint elrejtik az implementációjukat más vonatkozások elől. Azonban a vonatkozások szétválasztásának van egy nagy hátránya is. Fenn áll ugyan is annak a veszélye, hogy nem látjuk át a teljes rendszert.

Szétszóródás és összekeveredés

Ahhoz, hogy a vonatkozások szétválasztását elérjük, kívánatos, hogy a vonatkozások egymástól függetlenül specifikáljuk, implementáljuk és minél később illesszük össze egy rendszerré. Az alap vonatkozások, az objektum orientáltságnak köszönhetően könnyen modularizálhatóak, ám a kereszt vonatkozások szétválasztása még az objektum orientált eszközökkel sem érhető el teljesen. Ugyanis a vonatkozások szétválasztásánál a szétszóródás (scattering) és összekeveredés (tangling) problémájába ütközünk.

Szétszóródás: Egyetlen vonatkozáshoz kapcsolódik és akkor jön létre, ha a vonatkozás implementációja több modulban történik meg, azaz egy feladat több programegységben is megjelenik. A szétszóródás egyrészt programrészek redundanciájához, de legalább egymáshoz nagyon hasonló programrészek létrejöttéhez vezet. Ez nehezíti a rendszer módosíthatóságát, illetve növeli a hiba lehetőségek számát, mind a fejlesztés, mind a további esetleges módosítások során. Másrészt, módosítások esetén szinte követhetetlen egy nagy szoftver rendszerben, hogy mely modulokban történt már meg a módosítás, melyekben nem, megtörtént-e már minden helyen.

Összekeveredés: Egyetlen modulra vonatkozik és akkor jön létre, ha e modulba több vonatkozást (ill. vonatkozások részeit) implementáljuk, azaz olyan kódrészek, melyek több, különböző vonatkozáshoz tartoznak, de egyetlen egységbe kerülnek. Az

összekeveredés nehezen olvasható és nyomon követhetetlen kódot eredményez. De ami még nagyobb probléma, hogy az ilyen modulok nagyon nehezen, sőt sokszor egyáltalán nem használhatók fel újra. Valamint nehézé válik annak átlátása, hogy az egyes modulok, milyen problémát implementálnak.

Követelmény

Követelmény alatt az aspektus orientált világban egy olyan feltételt vagy képességet értünk, mellyel a szoftvernek rendelkeznie kell ahhoz, hogy a szoftvert megrendelő (cég vagy magán személy, stb.) által előírt vagy elvárt dolgokat teljesíteni tudjon. Egy részletes leírás a program feladatairól.

Funkcionális és nem funkcionális követelmények

A követelmények lehetnek funkcionálisak és nem funkcionálisak. Funkcionális követelmény alatt annak a leírását, megadását értjük, hogy egy szoftver rendszernek mit kell tennie. Míg nem funkcionális követelmény alatt, olyan átfogó tulajdonságokat értünk, melyek a teljes szoftver rendszer minőségére vonatkoznak és a funkcionális követelmények implementálására hatást gyakorolnak.(Ilyen tulajdonságok például: megbízhatóság, gyorsaság, biztonság, fejleszthetőség, kezelhetőség, költség, stb.)

Nem funkcionális követelmények és a kereszt vonatkozások

A nem funkcionális követelmények és a kereszt vonatkozások között egyfajta rokonság létezik azonban nem használhatóak egymás szinonimáiként. Ha, e két fogalmat halmazként képzeljük el, akkor a két halmaz metszi egymást, de semmiképp nem lehet az egyiket a másik részhalmazaként felfogni.

Kereszt vonatkozások, melyek nem nem funkcionális követelmények: A kereszt vonatkozások általában olyan tulajdonságok, melyek több mindent magukba foglalnak, egy fajta fogalom, melyet a programban majd meg kell valósítani (pl.: biztonság). De a követelményeknek, így a nem funkcionális követelményeknek is ellenőrizhetőeknek kell lenniük a megrendelő részéről. Épp ezért, mint az a követelmények definíciójában is szerepel részletesnek kell lenni. Egy olyan dolgot, mint a szoftver biztonságossága pontosabban meg kell határozni. Épp erre szolgálnak a követelmények, hogy ezt az elvárást részletesen kifejtsék, megadják. Tehát, a követelmények épp az ilyen nem egyértelműen definiált vonatkozásokat pontosítják.

Kereszt vonatkozások, melyek nem funkcionális követelmények is: Vannak azonban olyan részletesen megadott kereszt vonatkozások, melyek már implementálhatóak és így egyértelműen ellenőrizhetőek, így ezek nem funkcionális követelménynek is tekinthetőek.

Nem funkcionális követelmények, melyek, nem kereszt vonatkozások: Vannak olyan nem funkcionális követelmények is azonban, melyeket nem lehet kereszt vonatkozásokká felbontani és így implementálni. Olyan megkötések, melyek nem magára a programra, hanem a fejlesztésre vonatkoznak (pl.: kezelhetőség, bővíthetőség, stb.), vagy pedig olyan kritériumok, melyek a szoftver által megoldandó probléma, különböző megoldási lehetőségei közti választáskor lényegesek (pl.: költség, a fejlesztés időtartama, stb.). Az ilyen követelmények ugyan egyértelműen nem funkcionális követelmények, de semmi esetre sem kereszt vonatkozások, hisz nem magára a programra, hanem a fejlesztésre vonatkoznak.

Implicit követelmények

Minden szoftver olyan követelményeket is teljesít, melyek a funkcionális vagy nem funkcionális követelményekben nincsenek, vagy nincsenek egyértelműen rögzítve. Itt olyan magától érthető dolgokról van szó, melyeket minden programnak teljesítenie kell. Olyan tulajdonságokról, melyet a megrendelő elvár anélkül, hogy külön kérné. Azaz a rendszerre vagy a fejlesztésre vonatkozó elvárásai a megrendelőnek, melyek nem kerülnek explicit megadásra. Az implicit követelmények is lehetnek funkcionálisak és nem funkcionálisak. Az implicit és explicit követelmények közti határ változó, függ a programtól és a megrendelőtől. Általában minden esetben változik, nagyban függve attól, hogy a megrendelő és a fejlesztő(k) mennyire tapasztaltak. Ha a fejlesztő szervezeten belül, már bevált és minden esetben megkívánt tulajdonságokkal is ellátják, valamennyi programjukat, akkor ezen tulajdonságok sem kerülnek be az explicit követelmények közé.

Az implicit követelmények, noha nem jelennek meg egyértelműen, nyomtatott formában, sőt talán még szóban sem kerülnek kinyilvánításra, mégis nagyban hozzájárulnak egy szoftver rendszer minőségéhez. A megrendelő és a majdani felhasználók elégedettségéhez. Éppen ezért mindig nagy figyelmet kell szentelni rájuk és megfelelő módon meg kell őket valósítani.

Aspektusok és Komponensek

Általános eljárások (Generalized Procedure, GP):

Általános eljárás egy olyan fogalom, mely az eddigi programozási paradigmákban (objektum és eljárás orientált) is megjelenik, magát az absztrakciós és dekompozíciós mechanizmusát jelenti ezen paradigmáknak. Közérthetőbben, az általános eljárások maguk az objektumok, metódusok, eljárások, függvények, stb. Az az eszköz mellyel a problémát, feladatot részekre bontjuk és megoldjuk. Az ilyen típusú nyelvekben általában csak egyféle felbontási eszköz áll rendelkezésre (pl.: Az objektum orientált nyelvekben, az osztályokra kell felbontanunk mindent). Erre vezethető vissza, az összekeveredés és szétszóródás problémája, melyet az eddigi paradigmák nem voltak képesek kielégítő módon kezelni.

Aspektus (Aspect), Komponens (Component), Modul

Az aspektus orientált programozás három legfontosabb eleme a concernek mellett, maga az aspektus, a komponens és a modul. Épp ezért mielőtt részletesebben tárgyalnánk róluk, definiálnunk kell őket a lehető legpontosabban.

-Modul: Modul vagy egység alatt utasítások egy összefüggő sorozatát értjük, mely névvel rendelkezik és más program részek által ezen a néven keresztül meghívhatók.

-Aspektus (aspect): Míg a vonatkozások a probléma (a szoftver rendszerrel szemben támasztott igények, elvárások) megadására szolgál, addig az aspektusok ezen problémák megoldásának eszközei. Egy rendszer tulajdonságot aspektusnak nevezünk, ha nem lehet egyetlen általános eljárásba zárni. Az aspektusok ezért kereszt vonatkozások implementációja, másrészt modulok. Mivel az aspektusok kereszt vonatkozások implementálására szolgálnak általában nem a program központi feladatkörének megoldására használjuk.

-Komponens: A komponensek nagyon hasonlítanak az aspektusokhoz, mivel ezek is a problémák megoldására valók. Azonban egy rendszer tulajdonságot, akkor nevezünk komponensnek, ha egyetlen általános eljárásba összefogható. Azaz, a komponensek az alap vonatkozások implementációja, és szintén modult alkotnak. Természetesen a komponensek az alap vonatkozások implementálására szolgáló eszközként a program központi feladatkörének megoldását szolgálják.

Fel kell azonban hívnunk a figyelmet arra, hogy vannak az aspektus orientált paradigmának olyan nyelvei is, melyek nem tesznek különbséget az aspektusok és a komponensek között. Ilyen a Hyper/J. Ez érthető is, hiszen ezen nyelv az alap és kereszt vonatkozások között sem tett különbséget.
Most pedig vizsgáljuk meg részletesebben az aspektusok és komponensek közti kapcsolatot, hiszen végül is aspektusokból és komponensekből épül fel maga a szoftver.
A komponenseknek és aspektusoknak többféle tulajdonsággal is rendelkezniük kell, melyek mindkét esetben összehasonlítási alapot nyújtanak, az aspektusok és komponensek különbözősége miatt.

Összetartozás

Az összetartozás, egy modul belső összefüggésének mértéke. Az érték egytől, mely gyengét jelent hétig terjed, mely az erős összetartozást jelenti. Programfejlesztés során a modulok minél erősebb belső összetartozására kell törekednünk, ugyanis az erős belső összetartozás megkönnyíti a modulok kezelhetőségét és újra felhasználhatóságát.
Szintek:

1. Véletlen összetartozásról beszélünk, ha egy modul olyan műveleteket végez el, melyeknek semmi köze sincs egymáshoz, azaz nincs összetartozás.
2. Egy modul logikai összetartozással bír, ha egymással valamilyen kapcsolatban álló tevékenységeket hajt végre, mely tevékenységek közül a hívó modul valamilyen módon válogathat.
3. Múlandó összetartozás esetén a modul olyan műveleteket végez el melyek átmeneti kapcsolatban vannak egymással.
4. Egy modul összetartozása procedúrális, ha olyan műveleteket hajt végre, melyek előre megadott lépések sorozatár vonatkoznak.
5. Közlemény alapú összetartozás esetén a modul procedúrális összetartozással bír és minden művelete ugyanazokon az adatokon operál.
6. Információ vonatkozású összetartozása van egy modulnak, ha olyan műveleteket hajt végre melyek külön be és kimenettel, valamint implementációval rendelkeznek és ugyanazon az adatokon dolgoznak. Ilyen összetartozással rendelkeznek az olyan modulok, melyek egy absztrakt adattípusnak az implementációi.
7. Egy modul funkcionális összetartozással bír, ha egy műveletet hajt végre, vagy egy feladatot teljesít.

Az aspektusok esetén a funkcionális összetartozásnak kell megvalósulnia, mivel egy aspektusnak pontosan egy műveletet kell végrehajtania, méghozzá az általa implementált kereszt vonatkozások feladatát.

Összekapcsolás

Az összekapcsolás annak mértéke, hogy két modul mennyire függ egymástól. Értéke egytől, mely erős összekapcsolást jelent, ötig terjed mely a gyenge összefüggésnek felel meg.
Szintek:

1. Tartalmi összekapcsolásról beszélünk, ha a modulok a másik adataihoz hozzáférhetnek.
2. Két modul között általános összekapcsolás van, ha mindkét modul ugyanazokhoz a közös globális változókhoz tud hozzáférni.
3. Irányítási összekapcsolásról beszélünk, ha a hívó modul irányítási adatok átadásán keresztül, közvetlenül irányítani tudja a hívott modult.
4. Bélyeg alapú összekapcsolás esetén a hívó modul a hívottnak egy adatstruktúrát ad át, melynek a hívott azonban csak egy részét használja fel.
5. Adat alapú összekapcsolásról beszélünk, ha a hívó modul a hívottnak csak azokat az adatokat adja át, melyekre szüksége van.

Aspektusok és komponensek között az adat alapú összekapcsolást kell megvalósítani, ugyanis így érhető el a legnagyobb függetlenség, mely nagyban segíti a könnyű újra felhasználhatóságot, illetve a kezelhetőséget.

Bezárás

A bezárás azt jelenti, hogy az adatokat és az ezen operáló műveleteket egy modulban helyezzük el. Azonban az aspektusok esetén a funkcionális összetartozást kell megvalósítani épp ezért, a bezárást, nem lehet minden esetben feltétlenül megvalósítani aspektusok esetén.

Adatrejtés

Az adatrejtés fogalma, azt jelenti, hogy az adatstruktúrákhoz bizonyos műveleteket adunk meg és más modulok csak ezeken az interfészeken keresztül férhetnek hozzá az adatokhoz. Ez azonban már teljesen megegyezik az aspektusok szemléletével és nem csak kívánatos, de szinte kötelező az aspektusok számára.

Levonhatjuk, tehát a következtetéseinket az aspektusok és a komponensekre vonatkozóan. Mind az aspektusok, mind a komponensek modulok. A komponensek egy feladatot, egy alap vonatkozást valósítanak meg, míg az aspektusok a kereszt vonatkozások implementálásai. Ebből következően a komponensek feladat specifikus eszközök. Az éppen létrehozandó szoftver rendszer alapvető funkcióit megvalósító modulok. Nagy a valószínűsége annak, hogy a komponenseket semmilyen más szoftverben nem fogjuk tudni és valószínűleg nem is akarjuk majd felhasználni, hisz kifejezetten az adott problémához hoztuk őket létre. (Természetesen az erős összetartozás és a gyenge összekapcsolás mellyel a komponenseink bírnak, lehetővé teszik más rendszerekben való felhasználásukat is. Azonban ilyesmire általában nincs szükség.) Nem így az aspektusok, az aspektusok crosscutting concernek megvalósulásaként általános, egy szoftveren belül is sokszor és több helyen végrehajtandó feladat végrehajtásáért felelnek. Egy aspektus annál jobb, minél általánosabban valósítja meg a feladatát. Ideális aspektusról beszélhetünk, ha az aspektust tetszőleges szoftver rendszerhez kapcsolva átalakítások nélkül képes elvégezni a feladatát az új rendszeren belül is.

Kapcsolódási pont (Join point) és Szövés (Weaving)

Elértünk az utolsó két olyan fogalomhoz melyet definiálnunk kell, ahhoz, hogy az aspektus orientált paradigmához tartozó sokszínű programozási nyelvek közül választani tudjunk.
Az egyik ilyen fogalom a belépési pont.

Kapcsolódási pont: Egy interfész a programban, ahol a modulok egymással kapcsolatot tudnak teremteni. Kapcsolódási pont kisebb megszorításokkal a program tetszőleges azonosítható pontja lehet. (pl.: metódushívás, értékadás, stb.) Egy kódba több kapcsolódási pontot is elhelyezhetünk, de nem kell mindet felhasználni. Nem csak statikus (a kódba beépített) kapcsolódási pontok adhatók meg, hanem dinamikus (a futási időben, a program aktuális állapotától függőek) is létre hozhatók. A kapcsolódási pontok segítségével dinamikusan anélkül, hogy a program kódját átírnák különböző műveleteket hajthatunk végre a kapcsolódási pont előtt vagy után. Így növelve kódjaink újra felhasználhatóságát és könnyebb kezelhetőségét. A különböző nyelvek, különböző kapcsolódási pont modellekkel rendelkeznek. Ezek a kapcsolódási pont kezelésében, létrehozásában is nagymértékben eltérhetnek ezért részletes tárgyalását a kapcsolódási pontoknak és egyéb hozzá kapcsolódó nyelv specifikus fogalmaknak csak a nyelvek részletes tárgyalásánál fogjuk mélyrehatóbban vizsgálni.

Szövés: A szövés, egy a kapcsolódási pontokhoz szorosan kapcsolódó fogalom. A kapcsolódási pontokhoz hasonlóan ez a fogalom is minden aspektus orientált nyelvben megjelenik. A szövés, az aspektusok, komponensek és a kapcsolódási pontok összerendelését, összekapcsolását jelenti. Ilyenkor áll össze egy egészé a program. Ennek az a nagy jelentősége, hogy egy szoftver rendszert egy időben különböző helyeken, különböző csoportok fejleszthetik és csak később kell az egészet egy rendszerré összeilleszteni. Valamint az egyes részek így szabadon leválaszthatók vagy újabbak hozzácsatolhatóak a rendszerhez, kód átírása nélkül. Mikéntje, szintén nyelv specifikus. Létezik statikus (fordítási időben történő), dinamikus (betöltési időben történő) valamint futtatási idejű szövés is. Ezeknek a nyelvre vonatkozó, részletezésére szintén a későbbiekben kerül sor.

Összefoglalás

Mielőtt rátérnék az egyes nyelvek részletes elemzésére, bemutatására, fontos, hogy az eddigi fogalmakkal tisztában legyünk. Ezen fogalmak minden nyelvben megjelennek valamilyen módon és ami még ennél is fontosabb látnunk kell az egymással való viszonyukat.

Egy szoftver rendszer fejlesztése azzal kezdődik, hogy a fejlesztő (cég vagy személy (ek)) megkapja a megbízótól a követelményeket (explicit funkcionális és nem funkcionálisak). Gyakran nem egyértelmű a megbízó által kapott leírás, hisz nem biztos, hogy minden esetben olyan emberrel van dolgunk aki valamilyen szinten szakember. A fejlesztő feladata, hogy szétválogassa az explicit funkcionális és nem funkcionális követelményeket és amint azt az implicit követelményeknél említettük ezek meghatározása is roppant fontos. Helyesen szétválasztott követelmények nagyban megkönnyítik a későbbi tényleges fejlesztést, valamint nagymértékben hozzájárulnak a szoftver minőségéhez. Ezután, a funkcionális követelményekből elkészíthetjük az alap vonatkozásokat, a nem funkcionális követelményekből pedig a kereszt vonatkozásokat. Az implicit követelményekből mind alap, mind kereszt vonatkozások is létre jöhetnek. A vonatkozások létrehozása a követelmények részletes, szakszerű megfogalmazása, gyakran valamilyen algoritmus-leíró nyelven való leírása, megtervezése a programnak, melyet később a komponensekben és aspektusokban lekódolunk egy adott nyelvre. Ezt az egyes komponensek és aspektusok összeszövése és legvégül a program tesztelése követi.

Az AspectJ, az AspectWerkz és a Hyper/J

Amint az a fenti cím is mutatja három aspektus orientált nyelvel fogunk foglalkozni a továbbiakban. Azonban mielőtt az egyes nyelvek részletes tulajdonságaira rátérnénk, egy példán keresztül bemutatjuk az aspektus orientált programfejlesztés első két lépését, a követelmények és vonatkozások meghatározását. Végül a példa feladatot megvalósítjuk az egyik nyelv segítségével, még részletesebben bemutatva az adott nyelv eszközeit.

Követelmények

Amint azt az előző fejezet összefoglalásában említettük, a fejlesztés azzal kezdődik, hogy megkapjuk a megbízó által megfogalmazott feladat leírást, melyből nekünk ki kell válogatnunk azon információkat, melyek szükségesek ahhoz, hogy precízen megfogalmazhassuk az explicit, illetve implicit követelményeket.

Tegyük fel, hogy egy közepes méretű könyvtár megbízásából kell egy szoftver rendszert elkészítenünk, mellyel a könyvtárban az adminisztrációt végezik majd. A könyvtár megbízottja a következő feladatköröket, elvárásokat támasztotta a készítendő rendszerrel kapcsolatban.

A könyvtárban kétféle dolgot kell adminisztrálni:

- Könyvek
- Olvasók

Az egyes csoportok jellemzői:

-Könyvek: A könyvek egymástól különböző (három betűből és három számból álló) leltári számmal rendelkeznek. Minden könyvnél szerepelnie kell a szerzőnek és a könyv címének. Valamint nyilván kell tartani, hogy az adott könyv ki van-e kölcsönözve, vagy nem.

-Olvasók: Minden olvasó a beiratkozáskor egy ötjegyű számból álló belső azonosítót kap, valamint a rendszernek tárolnia kell az olvasó nevét, címét (város, utca, házszám, irányítószám), valamint ha létezik az olvasó telefonszámát. Emellett az olvasó által kikölcsönzött könyveket (azonosító, szerző, könyv címe), a kölcsönzés dátumát, illetve a lejárat dátumát.

A fenti csoportokon végezhető műveletek listája:
A könyvek listájára lehessen felvenni új könyvet, létező könyv adatait lehessen módosítani illetve törölni, valamint lehessen azonosító, cím vagy szerző alapján rákeresni egy könyvre. Azonban az olvasók ne módosíthassák a könyvek jellemzőit, valamint ne is vihessenek be új könyvet a rendszerbe, vagy törölhessenek ki onnan. De a felhasználók is kereshessenek rá a könyvekre azonosító, szerző vagy cím alapján. Az olvasók nyilvántartásához csak a dolgozók férhessenek hozzá, olvasó semmilyen műveletet ne hajthasson végre. Az olvasók nyilvántartásában is lehetőséget kell biztosítani a keresésre azonosító, név vagy cím alapján, valamint lehessen új bejegyzéseket fel venni és módosítani a már meglévőket. A törlés művelete mind az olvasók, mind a könyvek nyilvántartásában csak a dolgozók számára lehet lehetséges. Két fontos művelet, melyeket a programnak tudnia kell a könyvek kikölcsönzésének és visszavételének a lehetősége.
A program azonban legyen egyszerű és könnyen kezelhető, mivel az olvasóink is használhatják (korlátozva). Közöttük pedig sok gyermek és idős ember is van, akiknek nagy része nem jártas a számítás technikában.

Követelmények meghatározása

Ezen az egyszerű kis példa program fejlesztésén keresztül fogjuk megismerni és kielemezni az általunk választott nyelvet. Az első lépés, a fenti leírás alapján a követelmények szétválogatása és meghatározása, valamint egyértelmű, szakszerű megfogalmazása.

A programunk egy adatbázis kezelő program lesz. Kétféle adat struktúrát fog kezelni: könyvek és olvasók. Az olvasók és könyvek táblák között egyértelműen 1 : N kapcsolatot kell kialakítani, azaz egy olvasóhoz tartozhat több könyvek táblabeli elem, vagyis kikölcsönözhet több, különböző könyvet. De egy könyvek táblabeli elem csak egy olvasó táblabeli elemhez tartozhat, azaz egy könyv egyszerre csak egy olvasó által lehet kikölcsönözve.

Az olvasók táblának öt oszlopa lesz: azonosító (egy ötjegyű szám, elsődleges kulcs), név, cím (a cím oszlop egy rekord lesz, melynek város, utca, házszám illetve irányítószám oszlopai lesznek), telefonszám és egy beágyazott táblában, a kölcsönzött könyvek táblában fognak szerepelni az olvasó által kikölcsönzött könyvek. Ez a tábla a könyvek tábla egy-egy sorát fogja tartalmazni.

A könyvek táblának négy oszlopa lesz: azonosító (három betűből és egy háromjegyű számból álló kód, elsődleges kulcs), szerző (a könyv szerzőjének a neve), cím (a könyv címe) valamint egy kölcsönözve oszlopa lesz, mely rekord típusú lesz. A rekordnak három mezője lesz. Kölcsönző (azon olvasó azonosítója, aki kikölcsönözte a könyvet), kezdet (a kölcsönzés dátuma), lejárat (a kölcsönzési idő lejárta). Ez a rekord két állapotban lehet: üres vagy be állított. Ha be van állítva, akkor a könyvet kikölcsönző olvasó adatait, és a kölcsönzés, illetve a lejárat dátumát mutatja, míg az üres állapot azt jelenti, hogy a könyv nincs kikölcsönözve, tehát kölcsönözhető.
Mindkét táblához léteznie kell karbantartó műveleteknek: beszúrás (egy új sor felvétele a táblába), módosítás (a tábla egy létező sorának módosítása), törlés (a tábla egy létező sorának törlése), keresés (a tábla egy sorának keresése valamelyik attribútum alapján).
Két speciális műveletre lesz még szükség: kölcsönzés (könyv kölcsönzésekor a táblák megfelelő mezőinek beállítása), visszavétel (könyv visszavételekor a táblák megfelelő mezőinek beállítása). Emellett biztosítani kell, hogy a program bizonyos funkcióit, csak az arra jogosultak használhassák.

Az egyes műveletek részletes leírása:
-Beszúrás: A beszúrás művelete mind az olvasó, mind a könyvek táblára lehetséges. Beszúrást csak dolgozó hatáskörű felhasználó végezhet. Új elem felvételekor a könyvek táblába csak az azonosítónak kell különbözni, a többi attribútum megegyezhet más elemek attribútumaival. Könyv felvételekor a kölcsönözve oszlop értéke nem adható meg, implicit módon üresnek kell lennie. A többi oszlop értéke nem lehet üres. Az olvasók táblába való új elem felvételekor az azonosítónak különbözni kell az összes többitől, valamint nem fordulhat elő olyan, hogy a név és cím mezők együttesen megegyezzenek egy már meglévő elem ugyanazon értékeivel. A kölcsönzött könyvek attribútum szintén nem adható meg beszúráskor, ezt is implicit üresnek tekintjük. Beszúráskor egyedül a telefonszám mező kitöltése nem kötelező.

-Törlés: A törlés művelete mind az olvasó, mind a könyvek táblára lehetséges. Törlést csak dolgozó hatáskörű felhasználó végezhet. A könyvek táblából csak olyan elem törölhető, melynek kölcsönözve mezője üres. Az olvasók táblából is csak akkor törölhető egy elem, ha a kölcsönzött könyvek mező üres.

-Módosítás: A módosítás művelete mind az olvasó, mind a könyvek táblára lehetséges. Módosítást csak dolgozó hatáskörű felhasználó végezhet. Mindkét tábla esetén az elsődleges kulcsként funkcionáló azonosító oszlopok értékei nem módosíthatóak. A könyvek tábla esetén a szerző és a cím mezők értékei módosíthatóak. Az olvasók tábla esetén a név, a cím rekord és a telefonszám mezők értékei módosíthatók. Név és/vagy a cím mező valamely részének módosítása csak akkor lehetséges, ha a beszúrás műveletnél említett előfeltétel teljesül, azaz nincs egy másik olyan elem, melynek név és cím mezői teljesen megegyeznének az új értékekkel.

-Keresés: A keresés műveletére minden felhasználó jogosult. A keresés történhet a könyv vagy az olvasók táblában. A könyvek táblában azonosító, szerző és cím alapján lehet keresni, míg az olvasók táblában azonosító vagy név alapján lehet keresni.

-Kölcsönzés: A kölcsönzés műveletét csak dolgozó hatáskörű felhasználó végezheti el. A kölcsönzés művelete mind az olvasó, mind a könyvek táblát módosítani fogja. A művelet egy létező olvasó, kölcsönzött könyvek beágyazott táblájához fog hozzáadni egy sort, mégpedig a kölcsönzendő könyvhöz tartozó sort a könyvek táblából. A hozzáadás előtt azonban, a műveletnek be kell állítania a könyvek táblában a megfelelő könyv sorának kölcsönözve mezőjének minden értékét.

-Visszavétel: A visszavétel műveletét csak dolgozó hatáskörű felhasználó hajthatja végre. A visszavétel művelete mind a könyvek, mind az olvasók tábla adatai módosítani fogja. Az adott azonosítójú olvasó sorának, kölcsönzött könyvek beágyazott táblájából, a megfelelő könyvet reprezentáló sort törli, illetve ezen könyv kölcsönözve mezőjét üresre állítja.

Miután megfogalmaztuk a követeléseket, egy laza besorolást végezhetünk rajtuk.

-Explicit követelmények:

- *-Funkcionális követelmények:* Maguk a műveletek fognak ide tartozni. A műveletek működése, tehát az, hogy lehessen beszúrni, törölni, keresni, stb.
- *-Nem funkcionális követelmények:* Idetartozik a műveletek köré csoportosuló egyéb műveletek, pl.: azonosítás, feltételfigyelés, stb.

-Implicit követelmények:

Ide olyan tulajdonságai tartoznak a készülő programnak, melyek nem egyértelmű műveletek, hanem kényelmi dolgok. Például: beszédes, egyértelmű menüpontok, kényelmes vezérlés, stb.

Vonatkozások

Miután megfogalmaztuk a követelményeket és elvégeztük azok csoportosítását egy az előzőnél részletesebb tervet készíthetünk a majdani programunkról. A vonatkozások megtervezése valójában a program pontos megtervezését is jelenti. Itt döntjük el, hogy mely műveletek alkotnak alap és melyek kereszt vonatkozásokat, illetve, hogy az egyes műveleteknek pontosan hogyan is kellene majd működniük a kész programban.

Induljunk el a felhasználó irányából. Egy alap vonatkozás feladata lesz a menü létrehozása és kezelése. Azaz, azon felület (interfész) kialakítása, melyen keresztül a felhasználó kezelni tudja a programot. Ez egy nagyon fontos eleme a szoftvernek. Lehet akár milyen tökéletes egy program belső működése, ha a felhasználói felület nehezen kezelhető, vagy nem egyértelműek a menüpontok, a teljes program minősége romlani fog. Épp ezért célszerű egy több lépcsős menürendszert kialakítani, melyben a majdani felhasználó könnyen kiigazodik.
A program indításakor a program megpróbálja betölteni a merevlemezről a könyvek és olvasók táblákat. Amennyiben ez nem sikerül, jelzi, majd folytatja a szokásos működését. Kérni fog egy jelszót, ezzel azonosítja a dolgozókat. Ha nem adunk meg vagy nem megfelelő jelszót adunk meg a program az „olvasók módban” indul el és ilyenkor a főmenü csak a keresés és kilépés menüpontokat tartalmazza majd. (a keresés műveletének részletes leírására az alábbiakban kerül sor csakúgy, mint a kilépés menüpontéra.)
Ha a megfelelő jelszót adta, meg a felhasználó számára az összes programfunkció elérhetővé válik a menü rendszeren keresztül.
A mi esetünkben a fő menü öt menüpontot fog tartalmazni:

-Könyvkölcsönzés
-Könyv visszavétel
-Karbantartás
-Biztonság
-Kilépés

Minden menüponthoz egy sorszámot rendelve a felhasználó egyértelműen ki tudja majd választani a kívánt menü pontot. A beszédes nevek pedig segítik a döntésben.
Az egyes menüpontok:

-Könyvkölcsönzés

Ez egy alapvető művelet a programrendszerben, így a mögötte álló művelet is egy alap vonatkozást fog alkotni. A lényege a műveletnek az lesz, hogy a program bekéri a kölcsönzéshez kapcsolódó adatokat:

-Ki kölcsönzi a könyveket? (Olvasó kiválasztása az olvasók táblából.)

-Milyen könyveket kölcsönöz ki az olvasó? (Egy vagy több könyv hozzárendelése az olvasóhoz a könyvek táblából. Minden olvasó rekordhoz tartozik egy kölcsönzött könyvek lista, melyben az olvasó által kikölcsönzött könyvek kerülnek adminisztrálásra, a kölcsönzés és a lejárat dátumával. Ez a két dátumot a rendszer implicit módon határozza meg. A kölcsönzés dátuma a rendszerbeli aktuális dátum lesz, míg a lejárat ideje, a kölcsönzés dátumától számított huszonegy nappal későbbi dátum lesz. (Azaz, a kölcsönzési idő három hét) Azonban a könyvek táblában is módosításokra kerül sor. A kikölcsönzött könyvrekordoknak a kölcsönözve mezője ezután a kölcsönző rekordjára fog mutatni. Jelezve azt is, hogy a könyv ki van kölcsönözve.

-Könyv visszavétel

A másik nagyon fontos művelet. Az előző művelet ellentéte. Egy olvasó által kikölcsönzött könyvek visszavétele. Hasonló módon az előző ponthoz itt is egy alap vonatkozást kell megvalósítania a könyv visszavétel műveletét. A vonatkozás adatokat fog bekérni, és ezek alapján módosítja a könyvek és olvasók táblákat:

-Először is ki kell keresni a kölcsönző olvasó rekordját a táblázatból. Amennyiben az olvasó azonosítója ismert könnyű dolgunk van, azonban, ha ez az adat nem áll a rendelkezésünkre, az egyik kölcsönzött könyv kölcsönözve mezőjének a segítségével azonosítható a kölcsönző.

-Ezután az olvasó kölcsönzött könyvek listájából ki kell törölni a visszahozott könyveket, valamint ezen könyvekhez tartozó könyvek táblabeli rekordokban null értékre kell állítani a kölcsönözve mezőt.

-A programnak, mikor töröl egy könyvet a kölcsönzött könyvek listából, vizsgálnia kell a lejárat dátumát, és ha az kisebb mint a mai dátum, akkor jeleznie kell a művelet végrehajtása után, hogy lejárt könyv került visszavételre.

-Karbantartás

A karbantartás menüpont egy újabb menüt fog megjeleníteni. Itt a felhasználó a következő menü pontokból válogathat:

-Beszúrás

-Törlés

-Módosítás

-Keresés

-Vissza

Itt is sorszámozzuk a menüpontokat és a megfelelő sorszám kiválasztásával tud majd a felhasználó választani a menü pontok között. Maga a karbantartás, mint különböző műveletek halmaza tekinthető kereszt vonatkozásoknak, azonban az egyes műveleteket tekinthetnénk, külön-külön alap vonatkozásoknak is.

-Beszúrás menüpont

A beszúrás menü kiválasztása után egy újabb menü fog megjelenni:

-Olvasók

-Könyvek

-Vissza

Ez a menü segítségével tudja majd a felhasználó kiválasztani, hogy az adott műveletet mely táblán óhajtja végrehajtani. Illetve a vissza menü pont segítségével, visszatérhet az előző menüpontba. Amennyiben az olvasók táblában szeretne a felhasználó egy új elemet bevinni meg kell majd adnia az olvasó azonosítóját, nevét címét és esetleges telefonszámát. Ezen adatok közül csak a telefonszám megadása nem kötelező. A program ellenőrzi, hogy van e már ilyen adatokkal rendelkező olvasó a táblában. Két olvasó azonos, ha az azonosítójuk azonos, vagy megegyezik a nevük és a címük. Így csak egyedi azonosítójú, egyedi névvel és címmel rendelkező olvasó kerülhet felvételre. (Természetesen a név vagy a cím mezők értékei külön-külön meg egyezhetnek, csak az együttes azonosság tiltott.) Az adatok bevitelét egyértelmű kérések formájában fogalmazza meg a program. Hiba esetén, pedig szintén érthető üzenetben fogalmazza meg a hiba okát és lehetőséget ad a korrigálásra.

Ha a felhasználó a könyvek táblába szeretne bevinni egy új rekordot. Akkor az előzőekhez hasonló módon, meg kell adnia a könyv azonosítóját, címét és íróját. Ezen adatok közül mindet meg kell adnia. A könyveket az azonosítójuk egyértelműen azonosítja, így a címe és szerzője két könyvnek megegyezhet, csak az azonosítójuk nem.

Hibás adat bevitele esetén, a program itt is jelez és kéri a hiba javítását.

-Törlés menüpont

A törlés menüpont kiválasztásakor is megjelenik a menü, ahol a felhasználónak el kell döntenie, hogy mely táblát szeretné manipulálni. Ezután, a megfelelő azonosító megadásával kijelöli a felhasználó a törlendő könyvet vagy olvasót, és ha lehetséges törlődik a hozzájuk tartozó rekord a megfelelő táblából. Törölni csak olyan rekordot lehet, melyre nincs hivatkozás. Azaz könyv esetén a kölcsönözve mező üres, illetve olvasó esetén a kölcsönzött könyvek listája üres. Amennyiben ez nem teljesül, akkor a törlés nem hajtható végre.

-Módosítás menüpont

A módosítás menüponton belül is döntenie kell a felhasználónak, hogy melyik tábla adatain szeretne operálni. Amennyiben a könyvek tábla egy rekordjának adatait szeretné módosítani, úgy egy újabb almenü felkínálja a lehetőségeket, miután meghatározta, hogy melyik könyv mezőjének az értékét szeretné megváltoztatni. Könyv esetén a szerző vagy a cím változtatható meg, az azonosító nem és a kölcsönözve mező értékén sem változtathatunk.

-A módosítás menüpontban a könyvek táblát kiválasztva, a program bekéri a könyv azonosítóját. Amennyiben, létezik ilyen azonosítójú könyv a könyvek táblában, a program felkínálja választásra az adott könyv rekord szerző és cím mezőjét.

-A felhasználó kiválasztva a módosítandó mezőt, megadja annak új értékét, és a régi érték felül íródik. Ezután a program, visszatér az előző menühöz. A mennyiben itt a vissza menüpontot választjuk a fő menübe kerülünk vissza.

Ha azonban a felhasználó az olvasók tábla egy elemét szeretné módosítani, akkor szintén egy menü segítségével választhatja ki az olvasót, akinek az adatait módosítani szeretné. Majd azt, hogy mely adatait szeretné megváltoztatni. Egy olvasó adatai közül csak a név, cím és telefonszám mezők módosíthatóak, azaz az azonosítót nem módosíthatjuk itt sem, valamint a kölcsönzött könyvek listájába se nyúlhatunk bele. Erre csak a visszavétel művelet jogosult.

- A módosítás menüpontban az olvasók táblát kiválasztva, a program bekéri az olvasó azonosítóját. Ha talál ilyen azonosítójú olvasót menüpontban felkínálja a név, cím és telefonszám mezőket módosításra.
- A felhasználó választhat a pontokból, megadhatja az új értéket, és visszatér a mező kiválasztó menühöz, így egymás után több mezőt is módosíthat.

  A módosítások rögtön adminisztrálásra kerülnek. A mező kiválasztó menüből visszalépve a fő menübe jutunk.

Az új értékek megadásakor is hasonló módon vizsgálni kell az adatokat, mint azt a beszúráskor tettük, azaz már létező adatok, vagy hibás adatok nem kerülhetnek be.

-Keresés menüpont

Ebben a menüpontban is el kell döntenie a felhasználónak, hogy a könyvek vagy az olvasók táblában szeretne-e keresni.

A könyvek tábla esetén, egy könyvre az azonosító, cím és név alapján is kereshetünk. Azonosító alapján, biztos, hogy csak egy könyvet találunk. Cím vagy név alapján ebben nem lehetünk biztosak, így a megjelenítésnek fel kell rá készülnie, hogy több képernyőnyi adatot kell megjelenítenie, oly módon, hogy a felhasználó el tudja azokat olvasni. A menyiben a program nem talál megfelelő rekordot, úgy hiba üzenettel visszatér a keresés menü almenüjébe.

Ha az olvasók táblában szeretne keresni a felhasználó, nagy valószínűséggel azonosító, vagy név alapján kíván majd keresni (Valószínűtlen a cím vagy telefonszám alapú keresés). Így ebben az almenüben, csak e két lehetőség közül választhat a felhasználó. A találati eredmények hasonlóan a könyvekhez, azonosító esetén legfeljebb egy rekordot ad vissza, de név esetén nagyon sok is lehet. Így itt is megfelelő módon, kell kilistázni a találati eredményeket. Sikertelen keresés esetén a program szintén hibaüzenettel, visszatér a keresés almenübe.

-Vissza menüpont

A karbantartás almenü utolsó menü pontja, hatására a főmenübe térhet vissza a felhasználó.

-Biztonság

Ezen menüpontban, a felhasználó, megváltoztathatja a dolgozókat azonosító jelszót. Ez a funkció is egy külön álló egységet alkot ezért az alap vonatkozások közé sorolható.

-A menüpontba lépve, a program bekéri a régi jelszót és ha az megfelelő, akkor bekéri az új jelszót is.

-Ha nem volt jelszóval védve a rendszer, akkor felszólítja a felhasználót egy jelszó megadására. Amennyiben ezt elutasítja a felhasználó, figyelmeztet arra, hogy a program nincs jelszóval levédve és visszatér a főmenühöz.

-Kilépés

A kilépés menüpontot kiválasztva a program befejezi futását, leáll. Fontos, hogy a program leállása előtt az olvasók és könyvek táblák mentésre kerülnek.

Bizonyos műveletekről, azoknak megvalósítását a későbbiekben fogjuk kifejteni. Ugyanis ahhoz, hogy adott nyelven tárgyalni tudjuk a program megvalósítását elengedhetetlen, hogy megismerkedjünk a nyelvel.

AspectJ

Bevezetés

Az AspectJ[2] megtervezésekor a tervezők célja az volt, hogy az aspektusok külön nyelvi elemmé váljanak. Az *aspect* szó épp ezért kulcs szóként szerepel a nyelvben. Ugyan úgy készíthetünk aspektusokat mint ahogy osztályokat készíthetünk a Jávában. Futási időben ugyan úgy léteznek az aspektusoknak példányaik, melyek saját attribútumokkal és metódusokkal rendelkeznek, mint az osztályok példányi.

Az AspectJ tervezői az aspektus nyelvi elemként való integrálásával szerették volna elérni, hogy hosszú távon is versenyképes legyen a nyelv az egyre újabb aspektus orientált nyelvekkel szemben.

Az aspektus orientált programozás igazán akkor válik egy hatékony eszközzé, ha a programozó túllépve az objektum orientált világ strukturált gondolkodásán áttér az aspektus orientált világ gondolkodás módjára. Ez azonban csak gyakorlással érhető el. Épp ezért fontos célkitűzés volt az AspectJ megalkotásakor, hogy a nyelv könnyen tanulható legyen, hisz szinte senki sem akar egy teljesen új nyelvet megtanulni. Így az AspectJ a Jáva nyelvre épül rá, oly annyira, hogy minden legális Jáva nyelven íródott program, egy legális AspectJ programnak is tekinthető. Épp ezért amit az ember már meg tanult Jáva nyelven azt tudja használni AspectJben is. Ami igazán új, az a gondolkodási mód. Ezt talán jól szemlélteti, ha párhuzamot vonunk a Jáva és AspectJ valamint a C és C++ programozási nyelvek között. Ahogy a C++-t a C kiterjesztéseként értelmezték és átvezette a C programozókat az objektum orientált világba, úgy az AspectJ is átjárót képez a Jáva objektum orientált világából az aspektus orientált világába.

Az AspectJ az osztályok mellet lehetővé teszi aspektusok létrehozását. Az aspektusok hasonlóak az osztályokhoz csak vannak olyan képességeik is melyekkel az osztályok nem rendelkeznek.Kísérlet történt a kezdeti időszakban arra is, hogy az AspectJ már ne tartalmazzon osztályokat csak aspektusokat, de ez túl nagy lépésnek bizonyult. A legtöbb programozónál nem talált támogatásra. Az AspectJ tervezői nem csak egy kísérleti példa programok fejlesztésére alkalmas nyelvet akartak létrehozni, hanem egy olyat, mely a valós világ követelményeinek megfelelően, fejlesztői munkára is alkalmas. Ehhez azonban szükség volt, egy stabil fordítóra mely részletes hiba üzeneteket ad, gyorsan fut, és minőségi futtatható kódot generál. Az AspectJ tervezőinek sikerült elérniük a céljukat és létrehoztak egy versenyképes, megbízható, modern nyelvet.

Az AspecJ nyelvi elemei

Az AspectJ a Jáva nyelv kiterjesztéseként készült. Egy igazán fontos, új elemet ad csak hozzá a Jáva nyelvhez a kapcsolódási pontot (join point). Erre épülnek az AspectJ metszés pontok (pointcut) és tanács (advice) névre hallgató mechanizmusai. Ezen felül az AspectJ egyfajta nyílt osztály kompozíciót (open class composition) is engedélyez a típusközti deklarációként (inter-type declaration) ismert eljáráson keresztül. A metszés pontok, tanácsok és típusközti deklarációk szükség szerint kombinálhatóak és összefoghatóak egy moduláris egységben, egy aspektusban.

Kapcsolódási pont

A kapcsolódási pontok azonosítható pontok (események) a program futása során. Az AspectJ kapcsolódási pont modellje határozza meg, hogy milyen típusú kapcsolódási pontok hivatkozhatóak egy AspectJ programban.

A kapcsolódási modell a következő kapcsolódási pontokat definiálja:

-metódus vagy konstruktor végrehajtása
-egy tanács végrehajtása
-metódus vagy konstruktor hívása
-olvasási vagy írási hozzáférés egy attribútumhoz
-egy kivétel kezelő végrehajtása
-egy osztály statikus inicializálása
-egy objektum vagy aspektus inicializálása

Míg ezen események egy része statikus, a program szövegének a részeként azonosítható dolognak tűnik, fontos hogy megértsük a kapcsolódási pontok dinamikusak is lehetnek, azaz a program futása során kerülhetnek meghatározásra.

Metszés pontok

A metszés pontok kapcsolódási pontokat azonosító elemek. A metszés pont kifejezések az AspectJ primitív metszés pont kijelölőiből (designator) épül fel, melyeknek három alapvető fajtája van.

A metszés pont kijelölők első halmaza olyan módon választja ki, a kapcsolódási pontokat, hogy azok metódushívások vagy attribútum hozzáférések-e. A második csoportja a metszés pont kijelölőknek azon kapcsolódási pontokat választja ki, melyek egy meghatározott osztály példányihoz tartoznak, míg a harmadik halmazban azon kapcsolódási pontokat fogja kiválasztani a kijelölő, melyek egy adott csomag osztályának a kódjától származnak. Az

AspectJ tartalmaz metszés pont kijelölőket minden típusú kapcsolódási ponthoz. A kiválasztók a következők: `call`, `execution`, `get`, `set`, `handler`, `adviceexecution`, `staticinitalization`, `preinitalization` és `initalization`.

Az AspectJ kontextus függő metszés pont kijelölői a következők:

- `this (String)`: minden olyan kapcsolódási pontot kijelöl, ahol az aktuális futtató objektum példánya a String osztálynak.
- `target (XY)`: minden olyan kapcsolódási pontot kiválaszt, ahol a cél objektum (egy hívásnál például) az XY osztály példánya.
- `args (arg_nev)`: egy kiválasztott metódus formális paraméterlistájának az elemeit tudjuk elnevezni (kiválasztani) a segítségével. Lehetővé téve azoknak felhasználását.

Az AspectJ hatáskör alapú metszés pont kijelölői a következők:

- `within`: minden olyan kapcsolódási pontot azonosít, amelyekhez kapcsolódó forráskód a megadott hatáskörön (csomag például) belül van.
- `withincode`: azon kapcsolódási pontokat fogja kijelölni, amelyekhez kapcsolódó forráskód a megadott metódusok vagy konstruktorok halmazában van.
- `cflow`, `cflowbelow`: azon kapcsolódási pontokat fogja azonosítani, melyek egy bizonyos osztály példánya által kerültek meghívásra.

Az utolsó primitív metszés pont kijelölő melyet az AspectJ definiál az if kijelölő. Ez egy boolean kifejezést kap argumentumként. Az if kijelölő arra használható, hogy kiterjesszük az AspectJ kapcsolódási pont kiválasztásait. Általában arra használjuk, hogy egy logikai flag értékét teszteljük.

Az AspectJ egy fontos tulajdonsága, hogy a metszés pontok absztrakciója és kompozíciója is lehetséges. Az absztrakció a pointcut kulcsszóval lehetséges, ilyenkor ugyanis a programozó hozhat létre egy nevesített metszés pontot. Míg a kompozíció a nevesített és primitív metszés pontok &&, ||, ! operátorokkal való összekapcsolásán keresztül történik.

Tanács

A tanácsok egyszerűen, olyan program részek, melyek bizonyos metszés pont által kiválasztott kapcsolódási pontoknál végrehajtódnak. Minden tanácshoz hozzárendelődik egy metszés pont mely meghatározza, hogy mely kapcsolódási pontnál kell a tanácsnak lefutni.
Három alapvető tanácsot különböztetünk meg az AspectJ-ben:

-before advice: Ezen tanácsok a kiválasztott kapcsolódási pont előtt hajtódnak végre.

-after advice: Ezen tanácsok a kiválasztott kapcsolódási pont után hajtódnak végre.

-around advice: Ezen tanácsok a kiválasztott kapcsolódási pont helyett hajtódnak végre.

Az after advice tanács csoport három másik alrészre van felosztva:

-after returning advice: Ezen tanácsok csak akkor hajtódnak végre, ha a kiválasztott kapcsolódási pontban lévő esemény sikeresen végrehajtódott.

-after throwing advice: Ezen tanácsok csak akkor hajtódnak végre, ha a kiválasztott kapcsolódási pontban lévő esemény kivételt vált ki.

-after finally advice: Ezen tanácsok a kiválasztott kapcsolódási pontban szereplő esemény sikeres, vagy sikertelen (kivétel kiváltásával végződő) végrehajtása esetén is végrehajtódik.

Az AspectJ leghatékonyabb tanácsai az around adviceok, mert ezekben eldönthető, hogy a kapcsolódási pont eseménye lefusson-e, vagy ne. Megadható, hogy ha igen, akkor mi és ha nem, akkor mi hajtódjon végre.

Típusközti deklaráció

Kereszt vonatkozások néha nem csak dinamikus kezelést (tanácsok) hanem statikus kezelést is követelnek. Az AspectJ típusközti deklarációs technikája lehetővé teszi a programozó számára, hogy egy aspektust ellásson más típusokon nyugvó metódus implementációval és attribútumokkal.

Aspektus

Az aspektusok az AspectJ-ben a modularitás egységei, melyekben a kereszt vonatkozások implementálhatóak. Az aspektusok az aspect kulcsszóval deklarálhatóak. Metódusokat és attribútumokat tartalmazhatnak (mint egy osztály), de ezek mellett még metszés pontokat, tanácsokat és típusközti deklarációkat is tartalmazhatnak.
Mint az osztályoknak, az aspektusoknak is saját állapotuk és viselkedésük van melyeket az

attribútumaik és metódusaik határoznak meg. Az aspektusok példányai implicit létrejönnek a futási időben. Alapértelmezés szerint minden aspektus típusnak egy példánya jön létre, de más aspektus életciklusok is megadhatóak. Ehhez meg kell adnunk egy metszés pont kifejezést, mely pertarget, perthis, percflow vagy percflowbelow szerint létrehoz egy aspektus példányt. Például a percflow használatakor, minden esetben, mikor egy új control flow létrejön egy kapcsolódási pontban, melyet a metszés pont kijelöl, akkor létre fog jönni egy új aspektus példány. Ha egyszerre több aspektusnak is van tanácsa egy adott kapcsolódási ponthoz, akkor a végrehajtási sorrend, nem meghatározott. A legtöbb program számára ez megfelelő, de ha végrehajtási sorrend is számít, akkor az AspectJ lehetővé teszi prioritás hozzárendelését az aspektusokhoz. Biztosított egy declare precedence konstruktor, melynek segítségével az aspektusokhoz rendelt prioritásokon keresztül meghatározható a kapcsolódási pontokban a tanácsok végrehajtási sorrendje. Először a before azután az after tanácsok hajtódnak végre, két before vagy after tanács között a prioritás lesz a döntő, ha létezik.

Szövés

Az AspectJ a statikus szövés mellet foglal állást, azonban az új változat lehetővé teszi a betöltési időben történő szövést is. E két típus mellet még egyféle szövési mód létezik a futtatási idejű szövés. A statikus szövés során a program részeinek összerendelése már fordítási időben megtörténik, így a létrejövő program futása során minden előre meghatározott módon működik. Előnye a program gyors futása, hátránya, hogy új elemek beillesztésekor, újra kell fordítani a programnak az új modulhoz kapcsolódó részét. A Betöltési időben történő szövés esetén a class fájlok betöltése során történik meg az összerendelés. Utána úgy működik a program mintha statikusan szőttünk volna. Előnye, hogy új modul könnyen beilleszthető és a program futási teljesítménye se romlik, azonban a betöltés nagy időt is igénybe vehet, valamint nagyobb erőforrás igénye is van ennek a megoldásnak, mint a statikusnak. A harmadik módszert az AspectJ nem használja, azért sem mert a futási idejű szövés még egy fejlesztési időszakban álló koncepció. Vannak ugyan már kész rendszerek de működésük nem megfelelő. Amint azt a neve is mutatja ebben az esetbe, a program futása közben megy végbe a szövés, nem a fordításkor, vagy a betöltéskor. Ez a módszer tenné lehetővé a legnagyobb szabadságot a modulok csatolása és eltávolítása terén és talán az aspektus orientált szemléletnek ez lenne a legmegfelelőbb. Azonban ezen módszer nagy erőforrás igényű és még a futási minőség is romlik. A módszer kiforratlansága miatt a program összeomlásával is számolhatunk.

Összefoglalás

Az AspectJ, tehát egy dinamikusan fejlődő nyelv. Azonban, már ma is használható fejlesztési nyelvként, noha még nem teljesen kiforrott. A legújabb változat AspectJ5, mely a Java 5-ös technológiájára épít, sok új eszközt biztosít és az előző verziók sok hibáját is kiküszöbölték. Egy könnyen tanulható, izgalmas, új nyelv ez melyről még biztos sokat fogunk hallani.

AspectWerkz

Bevezetés

Az AspectJ mellett a másik legelterjedtebb aspektus orientált nyelv, mely élő és használt nyelvnek számít. Az AspectJ- vel nagyon sok közös vonása van. Szintén a Jáva nyelvre épít, azt egészíti ki. AspectWerkz-ban[3] is legális programnak számít egy Jáva nyelven íródott program, azaz az AspectWerkz is az objektum orientált Jáva nyelvet egészíti ki aspektus orientált eszközökkel, így téve könnyebbé a programozók átállását az új paradigmára. Hasonló nyelvi elemekkel rendelkezik mint az AspectJ, de van köztük eltérés is. Míg az AspectJ szorosan ragaszkodik a Jáva nyelv által definiált szintaxishoz, addig az AspektWerkz bizonyos esetekben aspektus orientált megvalósítását is lehetővé teszi bizonyos Jáva nyelvi szintaxisoknak. Ennél nagyobb eltérést jelent, hogy az AspektWerkz nagy hangsúlyt fektet a betöltési időben végbemenő szövésre, míg az AspectJ a statikus szövést preferálja.

Azonban a két nyelv közt szoros kapcsolatot áll fent, amit mi sem mutat annál jobban mint hogy, 2005. januárjában a két nyelv fejlesztői aláírtak egy megállapodást arról, hogy közösen létrehoznak egy aspektus orientált nyelvet, mely a már meglévő két nyelv erősségeit fogja egyesíteni. Az elhatározást tett követte és 2005. decemberében meg is jelent az AspectJ 5-ös verziója, melyet @AspectJ-nek is hívnak. Az új nyelv a közös munka gyümölcseként létrejött nyelv, mely sok új eszközt tartalmaz. Többek között teljesen kompatibilis a Jáva 5-ös változatával és nagyban épít is annak eszközeire, többek között a generikus programozásra, de megvalósult a betöltési időben történő szövés is. A nyelv még nagyon új ezért hibái lehetnek, de nagyon jó esélyekkel indul, hogy a jövő programozási nyelve legyen.

Az AspektWerkz nyelvi elemei

Aspektusok

AspektWerkz-ban minden java osztály lehet aspektus. Nem kell kiterjeszteniük semmilyen speciális osztályt vagy implementálniuk egy bizonyos interfészt, de kiterjeszthetik bármelyik osztályt. Az egyetlen megkötés, hogy az aspektus (osztály) ne rendelkezzen konstruktorral, eltekintve az implicit üres konstruktortól, vagy ha rendelkezik akkor a következő kettő közül valamelyikkel:

-az alapértelmezett argumentum nélküli konstruktor: csak akkor szükséges, ha az aspektusnak van másik, argumentummal rendelkező konstruktora

-egy olyan konstruktor melynek egyetlen argumentuma egy org.codehause.aspectwerkz.AspectContext példány: erre akkor van szükség, ha a futtató környezetből szeretnénk információt szerezni.

Az AspektWerkz elsőként a második konstruktort próbálja majd meg alkalmazni, ha ezt nem találja, akkor az alapértelmezett konstruktorral próbálkozik, ha azonban ez sem létezik, akkor kivétel fog kiváltódni.

Absztrakt aspektusok, aspektusok öröklődése

Mivel az aspektusok tiszta java osztályok, lehetőség van absztrakt aspektusok definiálására is, melyeket felhasználhatunk más, az absztrakt aspektus elemeit öröklő aspektusok létrehozásához. Az aspektusok öröklődése egy az egyben megegyezik az osztályok öröklődésével. Absztrakt aspektust úgy deklarálhatunk, hogy megadjuk az abstract kulcsszót. Az öröklődés mint egy átlagos osztályban az extends kulcsszóval történik.

Mindazonáltal bizonyos szabályokat figyelembe kell venni a tanácsok metódusként való implementálásakor. Habár úgy tűnik, hogy a tanács azáltal az osztály által hívódik melyben alkalmazásra kerül, mégis a tanácsoknak mindig publikusnak kell lenniük, soha nem lehet protected vagy private módosítót megadni. Csomag láthatóságú tanácsok ugyan létrehozhatóak, de nem javallott. Épp ilyen elgondolásból az aspektust megvalósító osztályoknak is publikusoknak kell lenniük. A csomag szintű láthatóság itt is engedélyezett, de nem tanácsos.

Tanács

Az aspektusokban a tanácsok egyszerű metódusok. A metódusoknak egy speciális szignatúrával kell rendelkeznie, hacsak nem kerül sor az args(), this(), target() kijelölők használatára a metszés pontban, amelyikhez a tanács tartozik.

-Around tanács esetén:

```
public Object <metódus neve>(StaticJoinPoint staticJoinPoint) throws Throwable signature
public Object <metódus neve>(JoinPoint joinPoint) throws Throwable signature.
```

-Before, After, After Finally, After Returning (típus) és After Throwing (típus) tanács esetén:

```
public void <metódus neve>() signature.
public void <metódus neve>(StaticJoinPoint staticJoinPoint) signature.
public void <metódus neve>(JoinPoint joinPoint) signature.
```

Metszés pont kijelölők használatakor (pl.: `args()`, `this()`, `target()`) a tanács alárendelődik a metszés pontnak. Így téve lehetővé a direkt hozzáférést a metódushoz, konstruktorhoz, attribútumokhoz és a formális paraméter listához. Magának a metszés pontnak is rendelkezni kell ilyenkor egy szignatúrával, és ekkor a metszés ponthoz kapcsolódó tanácsnak is tartalmaznia kell ezen paramétereket a szignatúrájában.
A tanács szignatúrája tehát a metszés pont szignatúrán alapszik és a kapcsolódási pontok nem használják.
A before és around tanácsok működése megegyezik az AspectJ-nél leírtakkal, azonban az AspektWerkz after tanácsaival nem egészen ez a helyzet. Az AspektWerkz 1.* változatában csak egyféle after tanács szerepelt. Ez úgy működött, hogy mindig lefutott a kapcsolódási pont után függetlenül attól, hogy milyen értékkel tért vissza egy metódus, váltódott-e ki vétel vagy sem.
Az AspektWerkz 2.x verzióban azonban átvéve az AspectJ after tanácsainak szemantikáját, három fajta after tanács közül választhatunk.

-after finnaly: Az egyes változat after tanácsa, mindig lefut, függetlenül a metódus sikerességétől, vagy más tényezőtől

-after returning [típus]: akkor kerülnek a tanácsban leírt utasítások végrehajtásra, ha a

metódus sikeresen tért vissza, nem váltott ki kivételt és a visszaadott típus megegyezik a tanács paraméterében megadott típussal. Ha nem adunk meg típust, akkor minden olyan hívás után lefut a tanács, mikor nem váltódik ki kivétel.

-after throwing [típus]: akkor fut le a tanács, ha a metódus kivételt váltott ki és a kivétel a tanácsban megadott típusú. Amennyiben nem adunk meg típust a tanácsban, tetszőleges típusú kivétel kiváltódása esetén lefut a tanács.

Kapcsolódási pont példányok

A JoinPoint osztály valósítja meg a kapcsolódási pont koncepciót, azaz a program futásában egy jól definiált pontra való hivatkozást és ebben a pontban oda-vissza hozzáférést biztosít egy API(Application Programing Interface)-hez. A StaticJoinPoint osztály az előbbinek egy egyszerűsített változata, az API hozzáférés nélkül. Így biztosítva gyorsabb futást.

A JoinPoint osztály példánya statikus információkat és RTTI-t (runtime type information / futási időbeli információk) biztosít a kapcsolódási pontról amelynél éppen a program a futása során jár. A StaticJoinPoint csak a statikus információkat tartalmazza. A statikus információk (a szignatúra) a szignatúra interfészen keresztül kerül átadásra a `getSignature()` metódus meghívásával. Az RTTI-t az RTTI interfészen keresztül kerül átadásra, a `getRtti()`, `getTarget()`, `getThis()`, `getCaller()`, `getCallee()` metódusok egyikének meghívásával.

A JoinPoint és StaticJoinPoint osztályoknak van egy proceed() metódusa, mely az around tanácsban kerül felhasználásra, vagy azért, hogy meghívjuk a következő tanácsot a hívási láncban, vagy pedig, hogy meghívjuk a cél kapcsolódási pontot, ha már nincs több tanács. A `proceed()` metódus a kapcsolódási pont mögött álló metódus visszatérési értékét adja vissza. Az around tanács esetén, így lehetőség van, hogy a metódus hívási helyére a `proceed()` metódus által szolgáltatott „helyes” értéket adjuk vissza, de „meghamisítható” az érték és egy tetszőleges érték is visszaadható.

Megjegyzés: (A JoinPoint osztálynak még sok más metódusa is van, azonban ezek részletes tárgyalása nem célja ezen dolgozatnak, de az AspektWerkz referenciában elolvasható.)

Statikus kapcsolódási pont, kapcsolódási pont, statikus kapcsolódási pont csatolás

A tanácsok metódusok az aspektusokban. A metódusok szignatúrájának meg kell felelnie néhány szabálynak, melyek a hozzájuk kapcsolódó ponitcutoktól függnek. Ezen, metszés pont

függő argumentumoktól eltérően egy tanácsnak lehetnek JoinPoint vagy StaticJoinPoint típusú argumentumai is, melyek nem jelennek meg a metszés pontokban. Az AspektWerkz korábbi verzióival ellentétben a 2.x verzó már nem csak JoinPoint típusú de StaticJoinPoint típusú paraméterek használatát is lehetővé teszi, azonban ezek használata teljesen a programozó kedvére van bízva. Jó ha megjegyezzük, hogy ha egy around tanácsban használni akarjuk a proceed() metódust szükségünk van egy StaicJoinPoint vagy JoinPoint példányra. Ha csak a StaticJoinPoint osztályt használjuk, akkor különböző optimalizációs lépésekre kerül sor, a legjobb futtatási teljesítmény eléréséhez, mivel RTTI hozzáférésre biztos nem kerül sor. Az AspectWerkz2.x.RC3-as verziója óta a StaicJoinPoint (és így a JoinPoint) osztály példányai is hozzáférnek az EnclosingStaticJoinPoint osztály példányaihoz az API getEnclosingStaticJoinPoint metódusán keresztül. Ez a call, get és set esetén lehet igazán, hasznos, hogy megtudjuk a csatolt statikus információkat, azaz hol következett be a call,get vagy set.

Szignatúra interfészek

A szignatúra interfészek arra használhatóak, hogy statikus információt szerezzünk a kapcsolódási pontról, ahol éppen a program futása jár. Az interfészek egy hierarchiát alkotnak, melyben igényeinknek megfelelően lépésről-lépésre, vagy célirányosan is haladhatunk. Utóbbi esetben azonban oda kell figyelnünk a megfelelő kaszttolásra. Egy szignatúra a getSignature() metódus meghívásának eredményeként kapható meg, egy StaicJoinPoint vagy egy JoinPoint példányból. Az interfészek a következők:

-org.codehaus.aspectwerkz.joinpoint.Signature

-org.codehaus.aspectwerkz.joinpoint.MemberSignature

-org.codehaus.aspectwerkz.joinpoint.CodeSignature

-org.codehaus.aspectwerkz.joinpoint.MethodSignature

-org.codehaus.aspectwerkz.joinpoint.ConstructorSignature

-org.codehaus.aspectwerkz.joinpoint.FieldSignature

-org.codehaus.aspectwerkz.joinpoint.CatchClauseSignature

RTTI interfészek

Az RTTI interfészek arra használhatóak, hogy RTTI-t szerezzünk a kapcsolódási pontról, ahol éppen a program futása jár. Az interfészek itt is egy hierarchiát alkotnak, melyben igényeinknek megfelelően szintén haladhatunk lépésről-lépésre, vagy célirányosan is. Utóbbi esetben azonban nem feledkezhetünk meg a megfelelő kaszttolásra. Egy RTTI, egy JoinPoint példány getRtti() metódusának meghívásával kapható meg. Az interfészek a következők:

-org.codehaus.aspectwerkz.joinpoint.Rtti

-org.codehaus.aspectwerkz.joinpoint.MemberRtti

-org.codehaus.aspectwerkz.joinpoint.CodeRtti

-org.codehaus.aspectwerkz.joinpoint.MethodRtti

-org.codehaus.aspectwerkz.joinpoint.ConstructorRtti

-org.codehaus.aspectwerkz.joinpoint.FieldRtti

-org.codehaus.aspectwerkz.joinpoint.CatchClauseRtti

Szövés

Az AspectWerkz az AspectJ-vel ellentétben már a kezdetektől fogva lehetőséget adott a statikus és a betöltési időben történő szövésre is. Az AspectJ 5-ös verziójában lévő betöltési idejű szövés is az AspektWerkz szövési módszerén alapul és a két szervezet együttműködésének zászlaja alatt jött létre. Az AspektWerkz fejlesztői azonban még tovább mennek és kísérleteket tesznek a futtatási időben történő szövés megvalósítására is, több-kevesebb sikerrel.

Összefoglalás

Az AspectWerkz az AspectJ méltó ellenfele, azonban a fejlesztők felismerve a két nyelv közti szoros hasonlóságot és az együtt működésben rejlő lehetőségeket, nem konkurensként, hanem társként működnek együtt. Így azok, akik az aspektus orientált paradigmával akarnak foglalkozni vagy akár már foglalkoznak, nem jelent nagy nehézséget a két nyelv közti átjárás. Az AspectWerkz is törekszik arra, hogy a fejlesztők érdekeit és elvárásait szemelőt tartva egy minden téren használható, sikeres programozási nyelvet valósítson meg.

Hyper/J

Bevezetés

A Hyper/J az IBM fejlesztéseként látott napvilágot. Egy rosszul dokumentált, nehézkes nyelvet sikerül alkotnia az IBM fejlesztőinek, épp ezért soha nem is került annyira előtérbe mint az AspectJ vagy az AspectWerkz. Azonban a Hyper/J teljesen az IBM fejlesztése, mely ugyan lehetővé teszi a Jáva nyelv használatát, de mint konkurens cég által fejlesztett nyelv, nem épít teljesen rá. A Hyper/J egy új szemléletmódot vezet be, mely ha megfelelően kidolgozzák akár sikeres is lehetett volna. Azonban az IBM 2001 januárja óta leállt a Hyper/J fejlesztésével így csak mint egy holt nyelvet tarthatjuk számon.

A Hyper/J nyelvi elemei

A Hyper/J az előző két nyelvel ellentétben teljesen más irányban ment el, és amint az a bevezetőben leírtuk, nem aratott túl nagy sikert, épp ezért csak néhány alapvető fogalommal ismerkedünk meg a Hyper/J világából

A Hyper/J felfogás szerint a program fejlesztés lépései:

1. Vonatkozások felosztása modul és rendszer szintű vonatkozásokra.
2. Implementálni kell a modul szintű vonatkozásokat osztályokban, míg a rendszerszintűeket aspektusokban (ez történhet szintén osztályokban).
3. Szövés

Vonatkozások több dimenziós szétválasztása

A Hyper/J technológia alapköve. Ezen új szemléletre épül az egész nyelv, és épp emiatt különbözik annyira minden más nyelvtől.

Előnyei:

-igény szerinti remodularizáció

-a fejlesztők összeválogathatják a legmegfelelőbb vonatkozásokat

Fogalmak:

Hipertér (hyper space): Több dimenziós vonatkozás tér (mátrix)

Egység (unit): Egy nyelv szintaktikai konstruktora.

Hiperszelet (hyper slice): Vonatkozások komplett, deklarált halmaza.

Hipermodul: Hiperszeletek halmaza.

Hipertér:

Lehetővé teszik minden vonatkozás és dimenzió azonosítását minden időpillanatban. A Hypertér bezárja a vonatkozásokat. Hiperterek kezelik és azonosítják a vanatkozások közti kapcsolatokat és a beépülésüket.

Egység:

Egy szintaktikai konstruktor egy nyelven. Például: deklaráció, osztály, interfész

-Privát egység: metódus, lokális változó

-Összetett egység: osztály, csomag

Vonatkozások dimenziói:

Többféle vonatkozás van. Alap vonatkozások, melyek az osztályok lesznek. Keresz vonatkozások melyek aspektusok lesznek, illetve a jellemzőként megvalósuló implicit követelmények. Ezeket együttesen nevezzük a vonatkozások dimenzióinak.

Egy hipertér dimenziói nxn-es mátrixba vannak elrendezve. Minden tengely a vonatkozások egy dimenzióját reprezentálja. Minden elem a mátrixban ezen dimenzió egy vonatkozását azonosítja. Így minden dimenziót aszerint definiálunk, hogy mire vonatkozik. Egy egység megmutatja az összes olyan vonatkozást melyhez kapcsolódik, de egy egység egy dimenzióban pontosan egy vonatkozáshoz kapcsolódik csak.

Hiperszelet:

A vonatkozás mátrixot azonosítja és elrendezi az egységeket és vonatkozásokat a vonatkozás dimenziói mentén, de nem támogatja a vonatkozások bezárását. Ez hiperszeletekkel történik. A hiperszelet vonatkozások olyan halmaza, melyek teljesek, azaz mindent deklarálnak amire szükségük van. (Metódusok és funkciók. Azonban a funkcióknak nem kell feltétlenül implementálva lenniük.) Így a hiperszeletek önállóak, nincsenek más hiperszeletekre ráutalva, de bizonyos funkciók elvégzéséhez azért lehet szükségük más hiperszeletekre.

Hipermodul:

A hiperszeletek építő kövek, melyeket összerakhatunk egy komplex rendszerré. Ez az összeillesztés történik a hipermodulok segítségével. A hipermodulok hiperszeletek halmazából és egybeépítési kapcsolatok (Integration Relationships) csoportjából áll, melyek

megmondják, hogy pontosan, hogy kell a hiperszeleteket összefűzni. A hipermodulok hiperszeletek is, hisz deklaratív komplettek, így hipermodulok egymásba ágyazhatóak. Az összefűzés történhet három különböző módon:

-Egyesítés név alapján: Egységeknek különböző hiperszeletekben azonos nevük van és kommunikálnak. Együtt egy új egységbe integrálódnak.

-Nem kommunikációs egyesítés: Az egységek véletlenül ugyanazzal a névvel rendelkeznek, de semmi más kapcsolat nincs köztük. Nem kerülnek összekötésre.

-Név szerinti felülírás: Az utolsó egység felülírja a megelőzőt, csak a metódusokra vonatkozóan, melyeknek neve megegyezik.

Összefoglalás

A Hyper/J egyik nagy eltérése a többi nyelvtől amint kiderül abból is adódik, hogy nem a Jáva technológiára alapoz, hanem azt mondja, hogy tetszőleges nyelven megírhatóak az egységek, így próbálva kiterjeszteni az alkalmazói kört. A Hyper/J nyelv azonban soha sem vált elterjedté, s olyan szintre sem emelkedett, hogy ténylegesen fejlesztésre lehessen használni. Azonban az új szemléletmód egy teljesen új világot nyit meg az aspektus orientált világban, a Jáva orientált programozói társadalom számára.

A példa program megvalósítása AspectJ nyelven

Miután megismerkedtünk az aspektus orientált nyelvek világával és áttekintettük bizonyos nyelvek legfőbb jellemzőit. Eljött az ideje, hogy a példa programunkat meg is valósítsuk az egyik, számunkra szimpatikus nyelven.
A programunk terve már az előtt elkészült, hogy az egyes nyelvekről akár csak említést is tettünk volna. Elég most eldöntenünk, hogy pontosan melyik nyelven is szeretnénk megvalósítani a programunkat, sőt nem kerülne túl nagy erőfeszítésbe egy másik nyelvre való átírása sem a programnak. Ha megfelelően készítettük el a feladat leírásából a követelményeket, majd a követelményekből a vonatkozásokat, akkor a program kódolásakor már nem kell túl sok tervezési problémával foglalkoznunk. Mi az AspectJ nyelv mellet foglaltunk állást, egyrészt azért, mert nagyon jól dokumentált nyelvről van szó, mely könnyen elsajátítható, másrészről pedig az egyik legelterjedtebb aspektus orientált nyelv.
Mielőtt azonban neki kezdenénk a tényleges programozásnak, jó ha látjuk magunk előtt a programot (vagy annak egy rész problémáját) teljes egészében. Ehhez a vonatkozások meghatározása című rész nagy segítséget nyújt a számunkra.

Foglaljuk tehát össze, hogy milyen elemekből (osztályokból) fog felépülni a programunk:
Könyveket és olvasókat kell nyilvántartanunk. Ezek, egymástól teljesen különböző, ám egymással kapcsolatban álló elemei a programnak. Mindkettőt egy-egy osztályban kell majd leképeznünk.

A könyvek osztály:
A könyvek osztálynak négy attribútuma lesz. Azonosító, szerző, cím és kölcsönözve. Mind a négy attribútumnak egy-egy osztályt feleltetünk meg, melyek tartalmazni fogják a megfelelő információkat, illetve különböző metódusokat valósítanak meg melyek egy része nyilvános, mert az adott osztály egy példányával való munkát segítik, (pl.: equals, toString, stb.) vagy a belső működéséhez szükséges, osztály láthatóságú metódusok. Ezen osztályok létrehozása egyszerű objektum orientált eszközökkel elvégezhető, nem igényel aspektus orientált eszközöket.
A könyvek osztálynak és az azonosító, szerző és cím attribútumokat képviselő osztályoknak a konstruktora speciális módon, a létre hozandó példány attribútumainak értékét a billentyűzetről fogja bekérni. Ez azért így történik, mert új könyv objektum létrehozását, csak felhasználó végezheti majd, illetve noha az azonosító, szerző és cím mezőket reprezentáló

osztályokat nem csak a könyvek osztály példányosításakor fogjuk használni, de létrehozásukhoz mindig felhasználói adat bevitelre lesz szükség.

Az olvasók osztály:

Az olvasók osztálynak öt attribútuma lesz, melyek közül négy általunk létrehozott osztály típusú (azonosító, név, cím és telefonszám), az ötödik attribútum, pedig egy láncolt lista lesz (kölcsönzött könyvek). Az azonosító, név, cím és telefonszám osztályok felépítése hasonló a könyvek osztályban felhasznált osztályainkhoz. Ezek is a megfelelő adatok tárolására, konstruktorukon keresztül azok billentyűzetről való bekérésére képesek, valamint különböző metódusokat valósítanak meg, globális vagy belső használatra. Ez az osztály is, a könyvek osztályhoz hasonlóan lényegében csak adat tárolásra és annak bevitelére képes. A létrehozandó adatbázisunk táblázatainak egy-egy rekordját képviselik. Épp ezért nem volt szükség aspektus orientált eszközökre megvalósításukkor, hisz valójában alap vonatkozásokról van szó.

Vizsgáljuk most meg azt az osztályt, mely lényegében felhasználja a könyvek és olvasók osztályokat.

A táblák osztály:

Az osztály elnevezése onnan ered, hogy ez az osztály fogja tartalmazni az adatbázisunkat képező két táblázatot, könyvek és olvasók. Mind két táblázat, melyet egy-egy lista fog reprezentálni, osztály szintű és nyilvános lesz. Ez a megoldás azért jó, mert így soha nem jöhet létre több példány a táblázatokból, valamint tetszőleges osztályból hozzáférhető, a csomagon belül. Az osztály további metódusokat biztosít a táblákon való manipuláláshoz. Így például beszúrás, törlés, keresés. Ezek az eljárások is a csomagon belül minden osztály számára láthatóak és az osztály szinten való deklarálás lehetővé teszi a példányosítás nélküli meghívásukat. Azonban nem szabad összetévesztenünk őket a programunk által megvalósítandó karbantartó műveletekkel. Ezek a metódusok a táblák osztály szerves részét képezik, ha úgy tetszik a táblázathoz való hozzáférés interfészei. Így egy önálló, kompakt egységet alkotnak.

A load-save osztály:

A táblák osztály után célszerű megemlítenünk a Load_Save osztályt melynek feladata, mint azt a neve is mutatja betöltés és mentés lesz. Ezen osztály a program indításakor megpróbálja

majd betölteni a merevlemezről a könyvek és olvasók táblázatot, amennyiben nem sikerülne, hiba üzenet mellett tovább folytatódik a program futása, üres táblázatokkal. Ezt a műveletet a betölt metódus hajtja végre. A másik metódusa az osztálynak a ment, mely a program leállítása előtt elmenti a táblákat a háttértárolóra. Mindkét osztály a java perzisztens eszközeit használja fel. Ezen technika lényege, hogy bonyolult osztály szerkezeteket nem kell szöveges vagy bináris fájlba leképeznünk, majd betöltéskor explicit visszaalakítanunk az adatokat. Lehetőség van az osztály példányok mentésére majd betöltésére. Így a teljes lista kimenthető, illetve betölthető anélkül, hogy nekünk gondot kellene fordítanunk az adatok háttértárra való leképezésének mikéntjére. Ezen műveletről eddig még szó sem volt, ez egy kimondatlan implicit követelményből származó vonatkozás. Hisz alapvető dolog egy adatok kezelésére szolgáló programnál, hogy eltudja menteni az adatokat és szükség esetén betudja tölteni azokat.

A kölcsönöz és visszavesz osztályok:

Ennek a két osztály egy-egy alap vonatkozás felel meg. Feladatuk, mint az a nevük is mutatja metódusaikon keresztül a könyvek kikölcsönzésének, illetve visszavételének adminisztrálása. Mindkét eljárás egyszerű, objektum orientált eszközökkel megvalósítható. Működésük lényege a vonatkozásokról szóló fejezetben elolvasható.

A security osztály:

Arról, hogy csak az arra jogosultak használhassák a program bizonyos funkcióit a security osztály metódusainak segítségével gondoskodunk. Az osztálynak két csomag láthatóságú metódusa van, a többi belső eljárás, ez a két metódus működéséhez szükséges. Az egyik nyilvános metódus, a jelszó kér metódus. Ennek feladata, hogy bekérjen a jelszót és eldöntse, hogy az a Security.dat fájlban tárolt jelszóval megegyezik-e. Ha nem létezik Security.dat állomány, fel ajánlja azt a lehetőséget, hogy létrehoz egyet. Ez fog történni az első futtatás során. Ha ilyenkor mégis úgy döntünk, hogy nem adunk meg jelszót figyelmeztetést kapunk, de tovább fut a program. A metódus egy logikai értéket fog vissza adni, attól függően, hogy helyes volt-e a jelszó vagy nem. A jelszó vált elnevezésű metódus a jelszó megváltoztatására ad lehetőséget. Ez a művelet áll a biztonság menüpont mögött.

Karbantart osztály:

Ezen osztály feladata lesz a karbantartási feladatok vezérlése. Négy metódust tartalmaz beszúr, töröl, módosít és keres. Mindegyik metódus a megfelelő paraméterezés segítségével el

tudja dönteni, hogy melyik osztályon kell dolgozni. A felhasználói utasítások, illetve adat bevitelek segítségével pedig kiválasztja az egyes karbantartási műveleten belüli rész feladatot és a táblák osztály metódusaira támaszkodva elvégzi azokat.

Menü osztály:

Ezen osztály feladata a menü rendszer megvalósítása. Vagyis ezen osztály metódusai írják ki a képernyőre a megfelelő menüpontokat, értelmezik a felhasználói döntéseket és annak függvényében cselekszenek.

Két fajta főmenünk lesz, egy teljes, mely a dolgozók számára lesz, tehát a jelszót ismerők számára és egy csökkentett módként értelmezett, a jelszót nem ismerő olvasók számára, mely csak a keres és kilépés opciókat tartalmazza.

A fő menü metódus, mint azt neve is mutatja fő menüt írja a képernyőre és egy menüpont kiválasztásakor meghívja a szükséges eljárásokat. Ez a kölcsönzés, visszavétel, biztonság és kilépés menüpontok esetében valóba, csak egy-egy metódus hívás, hisz alap vonatkozásokról van szó. Nem így a karbantartással. Ez egy külön menüt nyit meg, ebből is érezhetjük, hogy itt összetett, egymással és a program más részeivel összefonódó műveletekről van szó, azaz kereszt vonatkozásról. A karbantart menüponton belül, bármely funkciót kiválasztva a program arra fogja kérni a felhasználót, hogy döntse el melyik tábla adatain szeretne operálni. A tábla választ metódus fogja ezt egyértelműen meghatározni. Azonban annak érdekében, hogy a kód átlátható, újrafelhasználható, kereszthivatkozásoktól mentes legyen, itt be kell vetnünk az aspektus orientált eszközöket, pontosabban az AspectJ eszközeit.

A metsző eljárások aspektus:

Amint azt tárgyaltuk az AspectJ lehetőséget ad java osztályok mellet, aspektusok létrehozására, melyek a kereszt vonatkozásokat kezelik le. A metsző eljárások aspektus két metszés pontot definiál, valamint két hozzá tartozó tanácsot, valamint két metódust.

Metszés pontok:

```
pointcut modosit_keres(String argumentum):args(argumentum) &&
 cflow(execution(* Menu.karbantart_menu())) && execution(*
 Menu.tabla_valaszt_menu(String));
```

Metszés pontot a pointcut kulcsszóval lehet deklarálni. Utána meg kell adni a nevét és egy esetleges paraméter listát. Ez a metszés pont paraméter listája lesz, melyen keresztül a kapcsolódási pontok metódusainak paraméter listájához férhetünk hozzá. A név és a

paraméter lista után jön egy kettős pont majd a kapcsolódási pont kijelölők. Az args kijelölő segítségével a zárójelben megadott változón keresztül tudunk majd egy metódus, megegyező típusú argumentumára hivatkozni. A metszés pont argumentumának a neve kerül ide. A dupla és jellel, amint azt már megbeszéltük a metszés pontok kompozíciójára nyílik lehetőségünk. A cflow kijelölő segítségével a hívási lánc alapján hivatkozhatunk egy eseményre. Itt, azon kapcsolódási pontot, pontokat keressük, melyek a menü osztály karbantart menü metódusa után hívódtak meg. Az execution kijelölő segítségével a tábla választ menü metódus futtatását tudjuk kiválasztani. Tehát ez a metszés pont azt a pillanatot azonosítja a program futásában, mikor a karbantartó menüből meghívódik a tábla választ menü metódus, és a metszés pont a metódus aktuális paraméterét is elérhetővé teszi. Tettük ezt mindazért, hogy a következő tanácsban meg tudjuk határozni, hogy mi is történjen.

```
after(String argumentum) returning(String
valasztas):modosit_keres(argumentum){
            switch(new Integer(argumentum).intValue()){
            case 1 : beszur(valasztas); break;
            case 2 : torol(valasztas); break;
            case 3 : modosit(valasztas);break;
            case 4 : keres(valasztas);break;
            }     }
```

Ez amint látszik ez egy after tanács, méghozzá az a fajtája, mely a metszés pont által kijelölt metódus sikeres lefutása után hajtódik végre. Az after kulcsszó után megadjuk a metszés pont paraméterét, azaz a kijelölt metódus aktuális paraméterét, majd a returning kulcsszó után elnevezzük, amennyiben szükségünk van rá a kijelölt metódus visszatérési értékét, így ezzel is tudunk dolgozni. Ez után kettőspont után megadjuk, hogy melyik metszés ponthoz, tartozik a tanács, majd egy blokkban, hogy mi történjen. Jelen esetben, a tábla választ menü metódus paraméterében, magával hozza, hogy melyik menüpontból hívták meg. Ezt ugyanis nem tudjuk metszés pontok segítségével kideríteni, mivel egy case utasításon belülről hívódik meg. De így hozzáférve a paraméteréhez, eltudjuk dönteni, a hívás casen belüli helyét, a visszatérési értékétől függően azt is tudjuk, hogy melyik táblán kell végrehajtania a műveletet, és az aspektus belső metódusinak segítségével, megfelelően felparaméterezve meghívhatjuk a karbantart osztály megfelelő metódusait. Így annak eldöntését, hogy melyik táblán, esetleg, melyik rekordon kell műveletet végrehajtani, már nem a karbantartó osztály metódusainak kell eldönteni. Ezt elvégzi helyette az aspektus, s a karbantartó osztályban csak a tényleges műveletet kell végrehajtani.

Még egy metszés pont és hozzá tartozó tanács van az aspektusban.

```
Pointcut csokkentett_menu_keres():cflow(execution(*
      Menu.fomenu_csokkentett_mod())) && execution(*
      Menu.tabla_valaszt_menu(String));
after() returning(String valasztas):csokkentett_menu_keres(){
      keres(valasztas);
}
```

A metszés pont szinte teljesen megegyezik az előzővel, csak itt most nincs szükségünk a tábla választ menü metódus paraméterére, valamint azt a lefutását szeretnénk kijelölni, mely a menü osztály főmenü csökkentett mód metódusában következik be. Azért lesz ez külön est, mert nem ugyan onnan hívódik meg a metódus mint az előző esetben, valamint a paraméterezés nem lenne egy értelmű, mivel a két menüben, nem ugyan az lenne. Noha megvalósítható lenne, hogy ne keljen külön metszés pontot létrehozni, de megérni nem érné meg, csak a program átláthatósága és újra felhasználhatósága romlana.

A kivételek aspektus:

Ez az aspektus, mint azt a neve is mutatja arra szolgál, hogy a program futása során esetlegesen fellépő, hibákat kezelni tudjuk.

```
pointcut ures_tablak(): execution(* Load_Save.betolt());
after():ures_tablak(){
      if(Tablak.Olvasok == null || Tablak.Konyvek == null){
            Tablak.init_Tablak();
      }}
```

A program tesztelése során kiderült ugyanis, hogy a Load_Save osztály betölt metódusában van egy kis hiba. Ugyanis előfordulhatott az, az eset, hogy a metódus nem tudott egy láncolt listát betölteni. Vagy valamilyen hiba miatt, vagy azért mert nem is került elmentésre a tábla, szintén hiba folytán, vagy pedig azért mert még soha nem futattuk le a init táblák eljárást a táblák osztályban. Ezt a több, különböző, hibaforrást, egyetlen rövid metszés pont és tanács segítségével letudjuk azonban kezelni.

Az üres táblák metszés pont egyszerűen a betölt metódus lefutását azonosítja. Az after tanács akkor fog lefutni, ha a metszés pont által meghatározott esemény sikeresen vagy sikertelenül lefutott, azaz minden esetben. A művelet amit elvégez, igen egyszerű. Ellenőrzi, a betölt metódus lefutása után, hogy sikerült-e betölteni a táblákat, legalább egy ürest. Így ha azt tapasztalja, hogy a táblák null referenciára vannak állítva, akkor inicializálja őket, mivel a tábla osztály metódusai nincsenek felkészítve a null referencia kezelésére.

A start aspektus:

Egy utolsó aspektust kell még megtárgyalnunk a start aspektust. Ezen aspektusban került elhelyezésre a main metódus, a fő program.

```
public static void main(String args[]){
            Load_Save.betolt();
            Security.jelszo_ker();
            Load_Save.ment();
            return;
      }

pointcut indul(): cflow(execution(public static void main(..))) &&
      !cflow(execution( * Jelszo_valt())) && (execution(*
      Security.jelszo_ker()));
after() returning(Boolean ok) : indul(){
      try{
            if(ok){
                  Menu.fo_menu();
            }
            else{
                  Menu.fomenu_csokkentett_mod();
            }
      }
      catch(IOException ioe){
            System.out.println(ioe.getMessage());
      }
}
```

A főprogram azért került egy aspektusba, mert így egyetlen osztály sincs kitüntetett szerepben, valamint egyikhez sem kapcsolódik a feladatköre, és az, hogy egymás után több különböző osztály metódusait hívja szintén amellett szóló érv, hogy egy aspektusban helyezzük el. A fő program megpróbálja betölteni a táblákat, ha nem sikerül, hiba üzenetet kapunk, de hála a kivételek aspektusnak, egy üres táblával tudunk dolgozni. Ezután bekérjük a jelszót. Itt lép közbe az indul elnevezésű metszés pont, mely beazonosítja a program futásába azt a helyet, amikor is a main metódusból meghívásra kerül a jelszó kér eljárás, de nem a jelszó vált metódusból. Azért fontos, hogy csak a main metódusban való meghívását jelöljük, ki mivel jelszó kér metódus más helyen való meghívásakor nem szabad, hogy lefusson az indul metszés ponthoz tartozó tanács. Ez egy after tanács lesz mely csak a jelszó

kér metódus sikeres lefutás után hajtódik végre. A tanács kezeli a metódus visszatérési értékét is, így könnyen eltudjuk dönteni, hogy megfelelő jelszót adott-e meg a felhasználó. Ha igen akkor a menü osztály fő menü metódusát hívjuk meg, az összes program funkciót elérhetővé tesszük. Amennyiben helytelen jelszót adott meg a felhasználó, csökkentett módban fog futni a program.

Összefoglalás

Sok ember számára rémisztő dolog valami újnak a kipróbálása, elsajátítása, a régi feladása. Vannak helyzetek mikor nem is éri meg. Most azonban nem ilyen helyzettel állunk szemben. Az aspektus orientál világ szemléletmódja az objektum orientált világéra épül, azt terjeszti ki. Olyan új eszközöket ad a programozók kezébe melyek segítségével könnyebben, gyorsabban lehet megbízható, a kívánalmakat teljes mértékben kielégítő programokat írni.
A témában való kis mértékű elmélyülés is elegendő, hogy megérthessük az új paradigma előnyeit és hála a nyelveket fejlesztő szakembereknek a már ismert technológiákra épülő, így könnyen tanulható programozási nyelvek közül választhatunk. Valószínű, hogy a közeljövőben az egész világon és hazánkban is egyre elterjedtebbé válnak majd az aspektus orientált programozási nyelvek. Kényelmesebbé válik a programozók munkája és az új eszközök olyan feladatok könnyű és hatékony elvégzését is lehetővé teszik, melyek korábban bonyolult megvalósítást követeltek, s ebből következően potenciális hiba forrást jelentettek.
Mindenki számára csak ajánlani tudom, hogy minél előbb sajátítsa el ezen programozási nyelvek használatát. Könnyebbé és jobbá téve ezzel mind a programozói, mind a felhasználói társadalom mindennapjait.

Irodalomjegyzék

Robert E. Filman, Tzilla Elrad, Siobhán Clarke, Mehmet Aksit: Aspect oriented software development, 2005, Person Education Inc.

Wikipedia: Aspect oriented programing, 2006 ,Wikipedia Foundation
http://en.wikipedia.org/wiki/Aspect-oriented_programming

Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Videira Lopes, Jean-Marc Loingtier, John Irwin: Aspect-oriented programing, 1997, Springer-Verlag
http://www2.parc.com/csl/groups/sda/publications/papers/Kiczales-ECOOP97/for-web.pdf

Jan Richter: Einführung in AspectJ, 2005, Universität Leipzig, Institut für informatik
http://ebus.informatik.uni-leipzig.de/www/media/lehre/seminar-javatools05/semtools05-richter-text.pdf

Prof. Dr. Jürgen Ebert: Der Hyper/J ansatz, 2005, Universität Koblenz, Landau
http://www.uni-koblenz.de/FB4/Institutes/IST/AGEbert/Teaching/SS05/Seminar/FolienSchmitz.pdf

[1]Klaus L. Greulich: Hyper/J, 2003, Universität Bonn, Institut für informatik
http://www.informatik.uni-bonn.de/III/lehre/seminare/SWT/WS2003/ppt/HyperJ.pdf

Sebastian Schied: AOP mit AspectJ, 2003, Universität Bonn, Institut für informatik
http://www.informatik.uni-bonn.de/III/lehre/seminare/SWT/WS2003/ppt/AspectJ.ppt

George Blaschke: AspectJ – AOP ind der Praxis, 2003, Friedrich-Alexander Universität
http://www4.informatik.uni-erlangen.de/Lehre/SS03/HS_AspectOS/Ergebnisse/AspectJ-Folien.pdf

[2]The AspectJ programing guide, 2002-2003, Xerox Corporation, Palo-Alto research center
http://www.eclipse.org/aspectj/doc/released/progguide/index.html

[3]Jonas Bonér, Alexandre Vasseur : AspectWerkz Dokumentation, 2002-2005, Free Software Foundation
http://aspectwerkz.codehaus.org/

Köszönetnyilvánítás

Ezúton szeretné megköszönni témavezetőmnek, Espák Miklósnak a segítségét és türelmét, mellyel munkámat segítette. Az ő közreműködése nélkül nem készülhetett volna el ez a dolgozat.