8. Tétel

a) Magasszintű programozási nyelvek II.

Absztrakt metódusok és osztályok

- osztályok: "abstract", már 1 absztrakt metódusnál is absztrakt osztály lesz, lehet nem absztrakt metódusa is, ős absztrakt osztály absztrakt metódusokat muszáj felülírni, nem lehet példányosítani, ha csak absztrakt elemeket tartalmaz, akkor Interface

- metódusok: nincs törzse, kötelező felülírni, késői kötést használ (a v. m.-eknek van törzse és nem kötelező felülírni)

Interface-ek, többszörös öröklődés

- olyan speciális osztályfajta, amely önmagában nem tárol információt, csak propertyket, metódusokat definiál

- kizárólag abstract elemeket tartalmaz (nincs metódus törzs)

- minden eleme publikus ezért nem kell jelezni

- nem tartalmazhat mezőt és osztályszintű elemet

- def. szinten sincs konstrukor vagy destruktor

- kifejtéskor nincs szükség override kulcsszóra

- egy interfész implementálhat interfészt, egy osztály bármennyi interfészt implementálhat

- IComparable, IEnumarable, ICollection

Kivételkezelés filozófiája és megvalósítása

- az if ágak nem elég jók erre

- futás közbeni fellépi hiba, amely megszakítja az utasítások normális végrehajtását

- pl. hiányzó fájl, bemeneti hibák, túlindexlés, implicit (automatikusan), explicit (programozó)

- a kivételkezelés az a művelet, amely során felkészítjük a programot az esetleges futási hibákra, hogy ne térhessen vissza velük

- Exception osztály (hibaobjektum elkapása)

- kritikus részeket try-catch-el védeni, try ág: kritikus blokk, catch ágakban: kül. hibakódok, catch törzsében oldjuk meg a hibákat, legspeciálisabbtól a legáltalánosabbig, finally ág ami mindig lefut

Generikusok használata és készítése

- Boxing: referenciatípusú változónak egy értéktípusú értéket adunk

- double d = 12.4; object o = d;

- Unboxing: a Boxing ellentetje, az object változó értékét megkapja a megfelelő értéktípusú változó

- double x = (double)o;

- Object osztály: minden osztály őse, típuskompatibilitás elve pipa (GetType(), ToString(), Equals(), GetHashCode())

- Jobb megoldás: Generikusok-> típussal paraméterezhető osztály, interfész, metódus stb.

- paraméter keretekén <> jelek, majd példányosításkor is

- "where T:" kulcsszóval lehet kikötést tenni, hogy csak akkor fogadjuk el az adott T típusparamétert, ha ősül választotta a : után megnevezett osztályt (pl. public T Maximum<T>(T a, T b, T c) where T: IComperable {...}

Delegate-ek és események

- memóriában: adat és kód, adatra referencia típussal, kódra delegatekkel hivatkozunk

- Delegate: Metódus referencia

- delegate kulcsszó: delegate int Operation(int a, int b);

- delegatre illeszkedés: ugyanolyan a visszatérési típus és a paraméterlista, mint a delegaté

(Operationre írunk pl. egy Addition-t, Divisiont)

- példányosítás: Operation op; értékadása: op = Addition; futtatása: op(132, -83);

- generikus delegate: delegate T Operation<T>(T a, T b); -> Operation<int> op;

- Callback metódusok: egy metódust delegate-ként átadunk egy másik metódusnak (3 rétegű alkalmazásfejlesztésnél)

- Események: eseményvezérelt programozás, "event"

- delagatek listája, azok egyszerre való meghívása

- az event meghívásakor az összes benne tárolt delegate lefut. delegate void LogProc(string message)event / LogProc log;

- Anonim delegatek: név nélküli, egyszer használatos delegatek

- Lambda kifejezésekhez vagy ha nem akarjuk önállóan használni a delagatet

Lambda kifejezések

- deklarítv irány, "=>", SQL C# szintaxis keverék, x => x.EndsWith("dinnye"), kódismétlés kiiktatása

- lambda kifejezések hátterében: Predicate<T>, Action<T,..T16>, Func<T1,..T16>

- LINQ: nyelvbe ágyazott lekérdezés (Where, Slecet, GroupBy)

b) Fordítóprogramok

A fordítóprogramok alapjai

- Szerkezete: Forráskód program->Source handler<->Compiler->Code handler>Tárgynyelvű program

- fordítóprogram: forrásnyelvi szövegből tárgykódot állít elő

- feladata: nyelvek közötti konverzió végrehajtása

- a fordítóprogram a beolvasás után: elvégzi a lexikálás elemzést, szintaktikus és szemantikus elemzést, előállítja a szintaxis fát, generálja majd optimalizálja a tárgykódot

- tehát magas szintű-ből alacsonyszintű tárgykód (assmebly vagy gép kód)

- hardveres és virtuális interpreter (értelmező) -> közben fordítási idő és futási idő

- Megjegyzés: Analízis: (Lex, szin. és szem.) Szintézis: (kódgenerálás és optimalizálás)

Lexikális elemző

- inputja a source handler outputja (egy karaktersorozat)

- reguláris nyelvet használ (Chomsky 3-as)

- feladata felismerni az adott nyelv lexikális elemeit, majd helyettesíteni azokat a szintaktikus elemző számára érthető jelekkel. (outputja a szintaktikus elemző inputja + egy hibalista lexikális hibákról)

- a karakter sorozatban meghatározza azokat szimbolikus egységeket, konstansok változók, kulcsszavak

- if a > 0 then x := a else x := 0 -> ebből szimbólumtábla alapján -> 10 001 50 002 20 stb..

Reguláris nyelvek

- Reguláris nyelvek: 3-as típusú-> A->a A->Ba (használata a lexikális elemzés) + nem engedi az S->epszilont, Véges automatákkal felismerhetők

szintaktikai elemzők (Rekurzív leszállás módszere, LR(k), LL(k) elemzők táblázatos elemzők működése)

- környezetfüggetlen nyelvet használ 2-es típus

- meg kell határoznia a szintaxisfát, gyökér, élek, elemek (azaz program levezetése)

- Balról jobbra elemzésnél két ismert módszer:

- alulról felfelé LR(k): levél elemekből indulva haladunk az S szimbólum felé

- felülről-lefelé elemzés LL(k): az S -től indulva építjük a fát. (A szintaxisfa levélelemeire az elemezett szöveg terminális szimbólumai kerüljenek)

- LR(k) és LL(k) különböző megközelítéseket alkalmaznak a szintaktikai elemzés során

- Rekurzív leszállás módszere használatos az LL(k)-nál: rekurzív függvényhívások segítségével próbálja előállítani a bemeneti karakterláncot a nyelvtani szabályok alapján (a nyelvtani szabályokat a prediktv táblázat tartalmazza->sorain a nemterm. szimbólumok, oszlopain a bemeneti karakterek)

szemantikai elemzés kérdései

- környezetfüggő nyelvet használ 1-es típus

- feladata ellenőrzi hogy logikailag és értelmileg is helyes legyen

- típusellenőrzés (azonos típusú operandusok), függvény visszatérési típusok

- változók deklarálása, hatókör érvényesítése, konstansok értékek, literálok

- nevezési konfliktusok

A program fordítás lépései

- Compiler forrásnyelvű programot kap inputként

- Lexikai elemzés: Karakterekre bontja a forráskódot, azonosítja a kulcsszavakat, azonosítókat, literálokat.

- Szintaktikai elemzés: Ellenőrzi a tokenek szintaktikai helyességét a nyelvtan alapján

- Szemantikai elemzés: Értelmezi a program jelentését, típusellenőrzést és változó deklarációk ellenőrzését végzi.

- optimalizál majd tárgykód küldés