14. Tétel

a) Programozási technológiák

Az osztály jellemzői: felület és megvalósítás

- hasonlít a rekord adattípushoz mert inhomogén, tartalmazhat eljárásokat függvényeket

- a valóság egy (megfogható vagy megfoghatatlan) darabkájának absztrakciója

- azonban lehet technikai is (lásd pl. a tervezési mintáknál)

- OOP kimondja a használatát az egységbezárással azaz a rekordokat és az azokon dolgozó metódusokat egybeolvasztjuk, méghozzá osztályokba

- az osztály mezőkből és metódusokból áll.

- lehetnek példányai, a példányokat objektumoknak hívjuk

- más megközelítésből az osztályoknak két jellemzője van:

- felülete és viselkedése vagy implementációja

- az osztály felületét a publikus részei adják

- a viselkedését a metódusainak implementációja határozza meg (pl. getNév() viselkedése, hogy visszaadja a mező értékét)

az objektum jellemzői: felület, belső állapot és viselkedés:

- három jellemzője: felülete, viselkedése és belső állapota

- Kutya kutya = new Kutya ("Bodri");

- objektum: kutya, a Kutya osztály egy példánya

- az objektum felülete megegyezik az osztályának a felületével, azaz a kutya objektum és a Kutya osztály

- az objektumnak több típusa lehet

- viselkedése: metódusainak implementációja adja (a program futása közben változhat)

- belső állapota: mezőinek pillanatnyi állapota határozza meg (a mezők változékonysága miatt, felfoghatjuk úgy is hogy a metódusok az állapot átmeneti operátorok)

- az interfésznek csak felülete van, absztrakt osztálynak részleges viselkedése (ha minden metódusa absztrakt akkor egyáltalán nincs)

Az OOP alapelvek megfogalmazása ezekkel a fogalmakkal

- Egységbezárás (osztályokba) (belső értékek megvédése)

- Öröklődés

- Többalakúság (Kutya kutya = new Vizsla("Frakk") ne felejtsük el az Objectet sem;

A GOF1 és a GOF2 tervezési alapelvek bemutatása

- GOF1: 1995 Gang of Four: ez egy fontos alapelv: "Programozz felületre implementáció helyett"

- akkor programozunk implementációra ha kihasználjuk hogy egy osztály hogyan lett implementálva

- Tehát, ha a kódunkban találunk olyan részt, amely egy másik osztály implementációjától függ, akkor az hibás tervre utal

- Ha implementációra programozunk, és ha megváltozik az osztály, akkor a vele kapcsolatban álló osztályoknak is változniuk kell. Ezzel szemben, ha felületre programozunk, és megváltozik az implementáció, de a felület nem, akkor nem kell megváltoztatni a többi osztályt.

- GOF2: fontos alapelv: "Használj objektum összetételt öröklés helyett, ha csak lehet"

- öröklődés: IS-A kapcsolat (Kutya, Gerinces) -> az osztály összes szolgáltatását megörökli -> átlátszó újrahasznosítás

- objektum összetétel: HAS-A kapcsolat (Kutya, Gerinc) -> csak az osztály egy példányára szerzek referenciát és azon keresztül használjuk a szolgáltatásait -> átlátszatlan újrahasznosítás

- több fajtája van: Aggregáció(Gitáros dead->gitár->temetik), Kompozíció(Gitáros dead->gitár->eldobják), Becsomagolás (átlátszó-> A karácsonyfa karácsonyfa

marad, akárhány díszt is teszek rá és átlátszatlan->a becsomagolt karácsonyfa nem marad karácsonyfa)

A stratégia részletes, illetve a viselkedési tervezési minták általános bemutatása

- Stratégia:

- algoritmus családok definiálása és innen választunk a módszereknél

- viselkedési formáink vannak és ezekkel ruházzuk fel

- röviden: ugyanazt csináljuk, de másképp

- Ha van egy változékony metódusunk, akkor azt érdemes kiemelni egy osztály hierarchiába, amire egy referencia mutat. A referencián keresztül azt a változatot érjük el, amelyikre szükség van.

- Viselkedési tervezési minták: A viselkedési minták öröklés helyett összetételt

alkalmaznak. (Állapot, Megfigyelő, Template Method, Stratégia)

Az egyke részletes, illetve a létrehozási tervezési minták általános bemutatása

- Létrehozási t. m.:

- feladatuk, hogy megszüntessék a sok new kulcsszóval ránk szakadó függőségeket

- pl. new Kutya() helyett kutyaGyár.createKutya()

- pl. Egyke, Prototípus, Factory Method

- Egyke, Singleton:

- csak egy példány lehet belőle

- nem lehet publikus konstruktora (privát, hogy ne lehessen példányosítani) (tehát a megoldás: osztályszintű metódusokuk használata)

- az osztályszintű getInstance() metódus mindenkinek ugyanazt a példányt adja vissza

- sima megoldás: prviate static Singleton uniqueInstance=null;

- szálbiztos megoldás: private static readonly Singleton instance = new Singleton();

A díszítő részletes, illetve a szerkezeti tervezési minták általános bemutatása

- Szerkezeti t. m.:

- azt mutatják meg hogy hogyan használjuk a gyakorlatban az objektum összetételt úgy, hogy az igényeinknek megfelelő objektum szerkezetek létrejöhessenek futási időben

(3 féle obj. összetétel)

- pl. Illesztő, Díszítő, Proxy/Helyettes

- Díszítő:

- az átlátszó csomagolás klasszikus példája

- az obj. összetételben szereplő mindkét osztály ugyanazon őstől származik, azaz ugyanolyan típusú

- ez azért kell mert a díszítőelemek gyakran változnak

- megvalósítás: absztrakt ős (pl. fenyőfák) ennek kétfajta gyermeke van, alap osztályok amiket díszíteni lehet és a díszítő osztályok

- az ős minden meglévő metódusát implementálni kell

- példányosítás: ŐsOsztály példány = new DíszítőN(...new Díszítő1 ( new AlapOsztály(())...);

b) Adatbázisrendszerek II.

Kurzorok: implicit és explicit kurzorok kezelése

- Cursorok: lekérdezési eredmények sorainak kezelésére

- Implicit kurzor: az Oracle automatikusan kezeli, nem kell explicit módon deklarálni vagy megnyitni, akkor használjuk amikor csak egyetlen sorra vagy adatblokkra van szükségünk

- Explicit kurzor: deklarálni kell, megnyitni, kezelni és bezárni, általában akkor hasznosak ha egy lekérdezés több soros eredményt ad vissza, amelyeket soronként kell feldolgozni

- létrehozása: CURSOR employee\_cursor IS -> itt valamilyen SELECT

- megnyitása: OPEN

- általában a cursor rekordjait FETCH-eljük valamilyen LOOP-on keresztül, a CURSORT a program végén mindig le kell zárni CLOSE

- kurzor attribútumok:

- %FOUND: Igaz, ha a FETCH sikeresen beolvasott egy sort

- %NOTFOUND

- %ROWCOUNT: a beolvasott sorok számát adja vissza

- %ISOPEN: igaz ha a kurzor nyitva van

kurzorattribútumok használatának lehetősége

- kurzorok működésének ellenőrzésében és kurzorok állapotának lekérdezésekor használjuk

- hibák kezelésében

- kurzor attribútumok:

- %FOUND: Igaz, ha a FETCH sikeresen beolvasott egy sort

- %NOTFOUND

- %ROWCOUNT: a beolvasott sorok számát adja vissza

- %ISOPEN: igaz ha a kurzor nyitva van

- %(ROW)TYPE: például v\_salary employees.salary%TYPE;

kurzorváltozók

- adatok kiolvasásában, manipulálásában és tárolásában segítenek

- rekord változók: egy adott rekord struktúráját tükrözik (pl. a visszaadott sorok adatait ilyenekben tárolhatjuk) -> v\_employee employee%ROWTYPE;

- skalárváltozók: egyetlen értéket tárolnak, amelyet a kurzorokból olvasunk ki -> v\_salary employees.salary%TYPE

- index változók: a kurzorok által visszaadott sorok számát tárolják (pl. kurzorlépések számának követésére, vagy ciklusban iterációknál) -> v\_counter INTEGER := 0;

Tranzakciókezelés, triggerek a PL/SQL nyelvben

- Tranzakciókezelés: egy sor műveletet automatikusan (együtt) hajt végre, mintha azok egyetlen lépésben történnének. A tranzakciók biztosítják, hogy az adatbázisban végrehajtott változtatások vagy teljes egészében megtörténnek, vagy egyáltalán nem történnek meg.

- 4 fő ACID tulajdonsággal rendelkezik:

- Atomi művelet->vagy teljes egészében megtörténik vagy egyáltalán nem

- Consistency->A tranzakciók végrehajtása előtt és után is konzisztensnek kell maradnia az adatbázisnak

- Isolation->Úgy futnak mintha egyedül futnának a rendszerben, függetlenül attól hogy más tranzakciók futnak-e

- Durability->A tranzakció befejezése (COMMIT) után a változtatások tartósak maradnak

- kapcsolódó fogalmak: BEGIN TRANSACTION, COMMIT, ROLLBACK

- Triggerek: Olyan tárolt eljárások, amelyek automatikusan futnak, amikor bizonyos események bekövetkeznek egy táblában (pl. INSERT, UPDATE, DELETE).

- csoportosításai:

- BEFORE, AFTER (mielőtt és miután..)

- Szint alapján: Sor szintű és Utószintű

Összetett adatszerkezetek (kollekciók) típusai, jellemzői, használatának lehetőségei és előnyei, hátrányai

- adatok csoportosítására és kezelésekre használják

- Kollekciók típusai:

- Vektortípusok (Associative Arrays): kulcs-érték párokat tárolnak, az index lehet szám vagy karakterlánc, nincs fix méretük dinamikusan nőnek vagy csökkennek

- Tömbtípusok (Nested Tables):

- Azonos típusú elemek sorozatát tárolják, fix vagy dinamikus méret, üres elemeket is kezelhetne

- Gyűjtőtípusok (VARRAYs): fix méretű tömbök, azonos típusú elemeket tárolnak, szekvenciális hozzáféréssel rendelkeznek, azaz csak a végéhez vagy a kezdetéhez lehet hozzáférni

- Előnyei: rugalmas méret, egyszerű használat, hatékony adatfeldolgozás

- Hátrányok: nagy memóriaigény (nagy méretű kollekciók esetén), teljesítménybeli korlátok (pl. rendezésnél lassabb lehet a kollekcióban, mint SQL parancsok használata esetén)