Szakterületi fogalomjegyzék (Glossary)

Absztrakt adattípus (ADT, abstract data type)

Az absztrakt adattípus vagy ADT olyan egység, amely több különböző típusú elemet tartalmaz és műveleteket definiál rajtuk. Az ADT a C rekord típusához hasonlít, de tartalmazhat műveleteket is, mint a C++ osztályok. Lényeges eltérés a C++ osztályokhoz képest, hogy az ADT végleges és nincs polimorfizmus. [0]

Absztrakt osztályok (abstract class)

Ősosztályok "közös nevezőre" hozzák leszármazottaikat: definiálnak egy interfészt (külső felületet), vagyis egy adat-, és metóduskészletet, amelyen keresztül a leszármazottak egységesen kezelhetők. Megvalósítás nélküli metódusokat absztrakt osztályok tartalmazhatnak (abstract módosítószó az osztályhoz, metódushoz). A fordító nem engedi meg, hogy abstract módosítószó szerepeljen private, final, static mellett (ezeket nem lehet felüldefiniálni). Absztrakt osztály nem példányosítható. Absztrakt osztály kiterjesztésekor nem kell minden függvényt implementálni (de ekkor a leszármazottnak is absztraktnak kell lennie). [1]

Application Programming Interface (API)

Application Programming Interface = Felhasználói Program Interfész
Szabványos és jól dokumentált függvények és eljárások halmaza, amiket a programozó a szoftver és hardver vezérlésére használhat. [2]

Osztály

Az osztály az objektum-orientált programozás egyik központi fogalma. Az adattartalmukban esetleg eltérő, de viselkedés szempontjából megegyező objektumokat tekintjük egy osztályba tartozóknak. Az objektum-orientált programozási nyelvek rendszerint jelzik is az egyes osztályokat. Ennek kulcsszava a legtöbb esetben a *class*. [3]

Kivétel

A kivétel olyan esemény, szélsőséges eset, amelyre fel kell készíteni a programot az algoritmus írásakor. A kivétel előfordulhat hiba vagy speciális, ritkán bekövetkezhető esemény esetén. Egy részét szemantikai hibák váltják ki. Operációs rendszer szinten a kivételkezelést a megszakítási rendszer végzi. Kiírja, ha hiba történt és kivágja a programot. Kivételkezeléskor az a cél, hogy az esemény kezelését át kell venni a rendszertől. A kivételes események megoldására többféle módszer is létezik a nyelvekben. Plusz paraméterekkel jelzi a program, hogy milyen speciális esemény következett be. Ebben az esetben rengeteg paraméterre van szükség, és nem kezeli le a hibát, csak jelzi azt, például Pascal. Minden speciális esemény bekövetkezésére elágazást építünk be a programba. A legtöbb esetben a rengeteg lehetséges elágazás miatt nem valósítható meg. Az eljárásorientált nyelvekben, de az informatika más táján is a PL/1 és az Ada filozófiájú kivételkezelő jelenik meg, például adatbázis kezelésnél az Ada filozófia. Erre rakódik rá az objektumorientált nyelvek kivételkezelése. [4]

Objektum

Az objektum az objektumorientált programozás egyik alapeleme. Az objektumokat általában információt hordozó, és azokkal műveleteket vagy számításokat elvégezni képes egységként fogjuk fel.

A programozási feladat terében egy elemnek, vagy összetevőnek felel meg, míg számítógépes reprezentációja egy összefüggő adatterületként, és a területet kezelő alprogramok halmazaként képzelhető el. [5]

Kliens-szerver modell

Másnéven ügyfél-kiszolgáló modell. Az ügyfél (kliens) valamilyen kéréssel fordul a kiszolgáló felé, amely a kért szolgáltatást nyújtja. A programozásban a főprogram-alprogram viszonynak feleltethető meg. A kommunikációt mindig az ügyfél kezdeményezi, sohasem a kiszolgáló. Az ügyfél általában egy alkalmazói program: ilyenek a levelezőprogramok, a böngészők, stb. A modell legnagyobb előnye az, ahogy a kéréseket a hálózatra kapcsolt kiszolgálók között elosztja. Tegyük fel például, hogy több száz felhasználó akar egyszerre elérni egy adott számítógépen található állományokat. Ha mindegyik felhasználó bejelentkezne a kiszolgálóra, és az ott található ügyfél-programot használná, akkor a kiszolgáló nem győzné teljesíteni a kéréseket (ez azt jelentené, hogy a program egyszerre több száz példányban futna a kiszolgálón). Az ügyfél-programoknak az elosztásával a kiszolgáló terhelése csökkenthető, és így a feldolgozás főleg a felhasználó gazdagépén történik. A kiszolgálóval a kapcsolat csak akkor épül ki, ha adatcserére van szükség. [6]

Többrétegű architektúra

A webes alkalmazások tipikusan többrétegű (minimálisan három) architektúrára építenek.

Front End = kliens oldali felhasználói réteg (általában egy web-böngészőben)

Middleware = szerver oldali prezentációs és logikai réteg (általában egy web-szerveren a megjelenítés és a logika)

Back End = hátsó szerver oldali nagykapacitású tároló (adatbázis szerver) vagy számoló réteg. [7]

MVC

Az MVC paradigma egy szoftvertervezési minta / architektúra, mely a főként a webes alkalmazásoknál elterjedt. A monolitikus szoftverarchitektúrákat leváltó, három rétegű csoportosításról van szó, amiben az alkalmazás logikáját, vezérlését és felhasználói felületét elkülönítve kezeljük. A szeparáció legfőbb előnye, hogy egymástól függetlenül, akár modulárisan kicserélhetővé válnak egyes kódrészek, biztosítva, hogy például a UI lecserélését az üzleti logika átírása nélkül, tisztán és fájdalommentesen el tudjuk végezni. [8]

Model

A Model réteg modellosztályok összességéből áll. Egy-egy modellosztály általában az adatbázis egy tábláját és az adattípuson végezhető műveleteket reprezentálja és valósítja meg,

de lehet az adatbázistól független bármilyen absztrakció is. Régi keretrendszerek elterjesztették azt a terminológiai hibát, ami miatt a modellosztályokat szokás modellnek is nevezni, pedig a kettő más fogalmat takar. A modell, mint réteg szerepel az alkalmazásunkban, modellből egyetlen egy van. Ha valaki "models" könyvtárban dolgozik vagy "modellekként" emleget valamit, bizonyára a modellosztályra gondol. [8]

View

A View -szerencsétlenül- az MVC második betűje. Azért "szerencsétlenül", mert a Model közvetlenül nem kommunikál a Nézettel. A Nézet gyakorlatilag a frontendet jelenti, legyen az HTML, vagy egy desktop alkalmazás GUI-ja. A view generálásához gyakran használnak különféle template managereket, amelyek rövidebb és átláthatóbb kóddal emelik be a Model-től közvetetten megkapott megjelenítendő változókat. [8]

Controller

A Controller (Vezérlő) végzi az összeköttetést az MVC két másik rétege között. A Controller felelős a felhasználói inputok feldolgozásáért. A feldolgozás úgy értendő, hogy ebbe az adatbázisműveletek és a típusreprezentációval, integritással kapcsolatos funkcionalitás már nem tartozik bele - ez mind a modell feladata. A Vezérlő továbbá bizonyos user inputra bizonyos válaszokat adhat, lekérdezheti az adatbázist a modellen keresztül, majd továbbadhatja a kapott információt a Nézetnek. [8]

Programtervezési minta

Az informatikában a programtervezési mintának (angolul Software Design Patterns) nevezik a gyakran előforduló programozási feladatokra adható általános, újrafelhasználható megoldásokat. Egy programtervezési minta rendszerint egymással együttműködő objektumok és osztályok leírása.[9]

MVP (Model View Presenter)

Ebben a változatban a modell nem a nézetet, hanem a megjelenítőt értesíti, ha változás történik. A megjelenítő lekéri az adatokat a modellből, feldolgozza, és megformázza a nézet számára. [10]

…

[0] http://nyelvek.inf.elte.hu/leirasok/Limbo/index.php?chapter=6

[1] https://github.com/rlegendi/ELTE-javagyak/blob/master/05-oo-adt/05-oo-adt.md

[2] Pluhár Gábor - Informatikai értelmező szótár (PANEM könyvkiadó)

[3] http://wiki.prog.hu/wiki/Oszt%C3%A1ly

[4] http://wiki.prog.hu/wiki/Kiv%C3%A9tel

[5] http://wiki.prog.hu/wiki/Objektum

[6] Vincze Tamás: Hálózati kislexikon

[7] Tarcsi Ádám, Horváth Győző - Webadatbázis-programozás

[8] http://tananyag.net/f/mvc

[9] Szabolcsi Judit: Szoftvertechnológia - Programtervezési minták (Design Pattern)

[10] Kusper Gábor, Radványi Tibor - Programozás technika