

Dr. Mileff Péter

## Szoftverfejlesztés

Tervezés

1

### Objektumorientált felbontás

- Egy objektumorientált architektúris modell
  - > a rendszert lazán kapcsolódó, jól definiált interfészekkel rendelkező objektumok halmazára tagolja.
- Az objektumok a többi objektum által biztosított szolgáltatásokat hívják.
- A felbontás objektumosztályokkal, azok attribútumaival és műveleteivel foglalkozik.
- Implementációkor az objektumok ezekből az osztályokból jönnek létre,
  - > és az objektum műveleteinek koordinálásához valamilyen vezérlési modellt alkalmaznak.

2

### Objektumorientált felbontás

- Az objektumorientált megközelítés előnye:
  - > az objektumok lazán kapcsolódnak,
    - így az objektumok implementációja változtatható anélkül, hogy az hatással lenne más objektumokra.
  - > A komponensek közvetlen implementációjának biztosítására objektumorientált programozási nyelveket fejlesztettek ki.

3

### Objektumorientált felbontás

- Hátránya:
  - > az objektumoknak explicit módon kell hivatkoznia a többi objektum nevére és interfészére.
  - > Ha a rendszerben interfész változtatás történt,
    - akkor a változtatást minden, a megváltozott objektumot használó helyen át kell vezetni.

4

## Funkcionált csővezeték

- Más néven **adattfolyam-modell**.
- **Lényege:**
  - > Funkcionális transzformációk dolgozzák fel a bemeneteket
  - > és hoznak létre kimeneteket,
  - > közöttük áramlanak az adatok és sorban előrehaladva kerülnek átalakításra.
  - > Minden feldolgozási lépés transzformációként valósul meg.
- A bemeneti adatok ezeken a transzformációkon mennek keresztül, míg végül átalakulnak kimeneti adatokká.

5

## Funkcionált csővezeték

- A transzformációk mind szekvenciálisan, mind pedig párhuzamosan végrehajthatók.
- Az adatok egyesével vagy kötegelve is feldolgozhatók.
- Az interaktív rendszereket nehéz csővezetékkelvű modell alapján megírni,
  - > míg az egyszerű szöveges be-, illetve kimenet modellezhető ily módon.

6

## Funkcionált csővezeték

- **Előnye:**
  - > támogatja a transzformációk újrafelhasználhatóságát,
  - > könnyen érthető és bővíthető új transzformációkkal,
  - > implementálható konkurens és szekvenciális környezetben egyaránt.
- **Hátránya:**
  - > Az egyes transzformációknak vagy egyeztetniük kell egymással az átadandó adatok formátumát,
  - > vagy a kommunikációban részt vevő adatok számára egy szabványos formátumot kell kialakítani.

7

## VEZÉRLÉSI STÍLUSOK...

8

## Vezérlési stílusok

- Ahhoz, hogy az alrendszerek rendszerként működjenek, **vezérelni kell őket**.
  - > hogy szolgáltatásuk a megfelelő helyre a megfelelő időben eljussanak.
- A strukturális modellek nem tartalmazznak ilyen elemeket
  - > ezért a rendszer kiépítőjének **az alrendszereket valamilyen vezérlési modellnek megfelelően kell szerveznie**.
- A vezérlési modellek az alrendszerek közötti vezérlési folyamatokkal foglalkoznak.

9

## Vezérlési stílusok

- > **1. Központosított vezérlés:** a vezérlés teljes felelősségét egyetlen alrendszer látja el,
  - amely beindítja és leállítja a többi alrendszert.
- > **2. Eseményalapú vezérlés:** ahelyett, hogy a vezérlési információ egyetlen alrendszerbe lenne beágyazva,
  - minden alrendszer válaszolhat egy külsőleg létrejött eseményre.
  - Ezek az események vagy más alrendszerekből, vagy a rendszer környezetéből származhatnak.
- A vezérlési stílusok kiegészítik a strukturális stílusokat.
  - > Minden korábban bemutatott strukturális stílust meg lehet valósítani központosított és eseményalapú vezérléssel egyaránt.

10

## Központosított vezérlés

- A központosított vezérlési modellben létezik egy kitüntetett alrendszer:
  - > a **rendszer-vezérlő**,
- A többi alrendszer végrehajtásáért felelős.
- A központosított vezérlési modellek **két** csoportba oszthatók:
  - > attól függően, hogy a vezérelt alrendszerek szekvenciálisan vagy párhuzamosan hajtódnak-e végre.

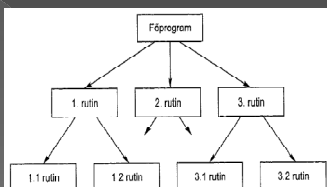
11

## Központosított vezérlés

- **1. A hívás-visszatérés modell:**
  - > A megszokott fentről le alprogram modellje,
  - > ahol a vezérlés az alprogram-hierarchia csúcsán kezdődik és alprogramhívások segítségével jut el a fa alsóbb szintjeire.
  - > Ez a modell csak szekvenciális rendszerek esetén alkalmazható.

12

## A hívásvisszatérés modell



- A vezérlés a hierarchiában magasabb szintű rutintól jut el az alacsonyabb szinten lévőhöz, majd visszatér a hívás helyére.

13

## A hívásvisszatérés modell

- A modell modulszinten használható a tevékenységek és objektumok vezérlésére,
  - > mert sok objektumorientált rendszerben az objektumok műveletei (metódusok) eljárásként vagy függvényként vannak implementálva.
- A modell merev és korlátozott természete egyben előny és hátrány is.
  - > **Erőssége** a vezérlési folyamat elemzésének és bizonyos bemenet esetén a rendszer válasza kiszámíthatóságának viszonylagos egyszerűsége.
  - > **Hátránya**, hogy a normálistól eltérő működés esetén használata kényelmetlen.

14

## Központosított vezérlés

### • 2. A kezelőmodell:

- > Konkurens rendszerekre alkalmazható.
- > Egy kijelölt rendszerkezelő rendszerkomponens irányítja a többi rendszerfolyamat indítását, leállítását, valamint koordinálja azokat.
- > A folyamat olyan alrendszer vagy modul, amely végrehajtható más folyamatokkal párhuzamosan.
- > Ez a fajta modell használható szekvenciális rendszerek esetén is.

15

## A kezelőmodell

- A rendszert vezérlő folyamat a rendszer állapotváltozói alapján dönti el:
  - > hogy az egyes folyamatokat mikor kell elindítani,
  - > illetve leállítani.
- Ellenőrzi:
  - > hogy a többi folyamat hozott-e létre feldolgozandó vagy feldolgozásra továbbküldendő információt.
- A vezérlő általában ciklikusan ellenőrzi az érzékelőket és folyamatokat,
  - > bekövetkezett-e valamilyen esemény vagy állapotváltozás.
- Éppen ezért ezt a modellt **eseményciklus-modellnek** is szokás nevezni.

16

## ESEMÉNYVEZÉRELT RENDSZEREK...

17

## Eseményvezérelt rendszerek

- ◉ Az esemény-vezérelt vezérlési modelleket külsőleg létrehozott események irányítják.
- ◉ Az eseményvezérelt rendszereknek sok fajtája ismeretes, beleértve a szerkesztőket,
  - > ahol a szerkesztő utasításokat felhasználói felület események jelzik.
- > **1. Eseményszóró modellek**
- > **2. Megszakításvezérelt modellek**

18

## Eseményvezérelt rendszerek

- ◉ **Eseményszóró modellek:**
  - > egy esemény minden alrendszerhez eljut,
  - > és bármelyik, az esemény kezelésére programozott alrendszer reagálhat rá.
  - > A vezérlési politika nincs beépítve az esemény- és üzenetkezelőbe.
  - > Az alrendszerek eldöntik, hogy mely eseményekre tartanak igényt,
    - az esemény- és üzenetkezelő pedig biztosítja, hogy ezek az események eljussanak hozzájuk.

19

## Eseményvezérelt rendszerek

- ◉ **Megszakításvezérelt modellek:**
  - > Ezeket kizárólag olyan valós idejű rendszerekben használják, ahol egy megszakítás-kezelő észleli a külső megszakításokat.
  - > Ezek aztán valamely más komponenshez kerülnek feldolgozásra.

20

## OBJEKTUMORIENTÁLT TERVEZÉS...

21

## Objektumorientált tervezés

- ◉ Egy objektumorientált rendszer egymással együttműködő objektumokból áll,
  - > amely az objektum saját állapotát karbantartja
  - > és erről az állapotról információs műveleteket biztosít.
- ◉ Az állapot reprezentációja privát,
  - > az objektumon kívülről közvetlenül nem hozzáférhető.
- ◉ **Egy objektumorientált tervezési folyamat az objektum-osztályoknak és az azok közötti kapcsolatoknak a megtervezéséből áll.**

22

## Objektumorientált tervezés

- ◉ Az objektumorientált tervezés az objektumorientált fejlesztés része
  - > a fejlesztési folyamat során objektumorientált stratégiát használunk:
- ◉ **1. Objektumorientált elemzés:** a szoftver objektumorientált modelljének kialakításával foglalkozik.
- ◉ **2. Objektumorientált tervezés:** a meghatározott követelményeknek megfelelő szoftverrendszer objektumorientált modelljének kialakítása.
  - > Az objektumorientált tervezés objektumai a probléma megoldásával kapcsolatosak.
- ◉ **3. Objektumorientált programozás:** a szoftvertervezési objektumorientált programozási nyelven történő megvalósítása. Pl.: C++, D, Java.

23

## Objektumorientált tervezés

- ◉ A különböző lépések közötti átmenetnek észrevehetőnek kell lennie,
- ◉ Mindegyik lépésben kompatibilis jelölésrendszert kell használni.
- ◉ Az objektumorientált rendszereket a más elven fejlesztett rendszerekkel szemben könnyebb megváltoztatni,
  - > mert az objektumok egymástól függetlenek.

24

## Objektorientált tervezés

- Egy objektum implementációjának megváltozása vagy új szolgáltatásokkal történő bővülése nem befolyásolhatja a rendszer többi objektumát.
  - > Ezért azt mondhatjuk, hogy az objektumok potenciálisan újrafelhasználható komponensek,
  - > mivel az állapotnak és a műveleteknek független egységbe zárásai.
  - > Ez növeli az érthetőséget és így a terv karbantarthatóságát is.

25

## Objektumok és objektumosztályok

- Ian Sommerville féle objektum definíció:
- Egy objektum egy állapottal és az ezen az állapoton ható, meghatározott műveletekkel rendelkező entitás.
- Az állapotot objektum-attribútumok halmazaként adjuk meg.
- Az objektum műveletei szolgáltatásokat biztosítanak a többi objektum számára.
- Az objektumok egy objektumosztály-definíció alapján jönnek létre.

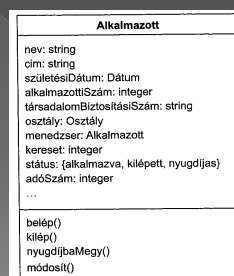
26

## Objektumok és objektumosztályok

- Egy objektumosztály definíciója
  - > egyszerre típuspecifikáció
  - > és egy objektumok létrehozására szolgáló sablon.
  - > Az adott osztályba tartozó objektummal kapcsolatos összes attribútum és művelet deklarációját tartalmazza.

27

## UML-ben megadott osztálydefiníció



- Az Alkalmazott objektumosztály

28

## Objektumok és objektumosztályok

- ◉ Az objektumok úgy kommunikálnak,
    - > hogy szolgáltatásokat kérnek más objektumoktól (meghívják azok módszereit).
    - > A szolgáltatás végrehajtásához szükséges információ és a szolgáltatás végrehajtásának eredményei paraméterként adódnak át. Pl.:
- ```
// Egy puffer objektum módszerének hívása, amely visszaadja a pufferben található következő értéket
v = Buffer.getNextElement();
// Puffer elemszámának a beállítása
v = Buffer.setElements(20);
```

29

## Objektumok és objektumosztályok

- ◉ Egy objektum „szolgáltatáskérés” üzenetet küldhet egy másiknak,
  - > akitől a szolgáltatást kéri.
- ◉ A fogadóobjektum:
  - > elemzi az üzenetet,
  - > azonosítja a szolgáltatást és az ahhoz kapcsolódó adatokat,
  - > majd végrehajtja a kívánt szolgáltatást.
- ◉ Ha a szolgáltatáskérések ilyen módon vannak implementálva, az objektumok közötti kommunikáció **szinkron**,
  - > azaz a hívóobjektum megvárja a szolgáltatás befejeződését (**soros végrehajtás**).
  - > A gyakorlatban a legtöbb objektum-orientált nyelvben ez a modell az alapértelmezett.

30

## Objektumok és objektumosztályok

- ◉ Az újabb OOP nyelvekben azonban, mint pl. a JAVA-ban vagy a D-ben, léteznek a szálak,
  - > amelyek megengedik a konkurens módon végrehajtható objektumok létrehozását
  - > és az **aszinkron kommunikációt** (a hívóobjektum folytatja a működését az általa igényelt szolgáltatás futása alatt is).
- ◉ Ezek a konkurens objektumok kétféleképpen implementálhatók:
  - > **Aktív objektumok vagy Szerverek.**

31

## Objektumok és objektumosztályok

- ◉ **1. Aktív objektumok:**
  - > önmaguk képesek belső állapotukat megváltoztatni és üzenetet küldeni,
  - > anélkül, hogy más objektumtól vezérlőüzenetet kaptak volna. (Ellentétük a passzív objektum.)
  - > Az aktív objektumot reprezentáló folyamat ezeket a műveleteket folyamatosan végrehajtja, így soha nem függesztődik fel.

32



## Objektumok és objektumosztályok

### 2. Szerverek:

- > az objektum a megadott műveleteknek megfelelő eljárásokkal rendelkező párhuzamos folyamat.
- > Az eljárások egy külső üzenetre válaszolva indulnak el és más objektumok eljárásaival párhuzamosan futhatnak.
- > Mikor befejezték a tevékenységüket, az objektum várakozó állapotba kerül és további kéréseket vár.

33

## Szerverek alkalmazásai

- ◉ A szervereket leginkább osztott környezetben érdemes használni,
  - > ahol a hívó és a hívott objektum különböző számítógépeken hajtódik végre.
- ◉ Az igényelt szolgáltatásra adott válaszidő megjósolhatóan,
  - > ezért úgy kell megtervezni a rendszert, hogy a szolgáltatást igénylő objektumnak ne kelljen megvárni a szolgáltatás befejeződését.
- ◉ A szerverek persze egyedi gépen is használhatók
  - > ahol a szolgáltatás befejeződéséhez némi időre van szükség
    - pl. nyomtatás és a szolgáltatást több különböző objektum is igényelheti.

34

## Aktív objektumok alkalmazásai

- ◉ Aktív objektumokat akkor célszerű használni,
  - > ha egy objektumnak saját állapotát megadott időközönként frissíteni kell.
- ◉ Ez valós idejű rendszerekben gyakori,
  - > ahol az objektumok a rendszer környezetéről információt gyűjtő hardvereszközökkel állnak kapcsolatban.

35

## Objektumok és objektumosztályok

- ◉ Az objektumosztályok egy generalizációs vagy öröklődési hierarchiába szervezhetők,
  - > az általános és a specifikus objektum-osztályok közötti kapcsolatot jeleníti meg.
- ◉ A hierarchiában a lejjebb található osztályok:
  - > rendelkeznek ugyanazokkal az attribútumokkal és műveletekkel, mint szülőosztályaik,
  - > de új attribútumokkal és műveletekkel egészítheti ki azokat,
  - > valamint meg is változtathatják szülőosztályaik attribútumainak és műveleteinek némelyikét.

36

## Objektumok élettartalma

- Egy objektum belső állapotát az attribútumainak pillanatnyi értéke határozza meg.
- **Perzisztens objektumok:**
  - > A háttértárolón azon objektumok adatait kell tárolni, amelyek élettartalma hosszabb, mint a program futási ideje.
- **Tranziens objektumok:**
  - > Azon objektumok, amelyek élettartalma a program futási idejénél nem hosszabb.
- Ha a program nagyszámú perzisztens objektummal dolgozik,
  - > érdemes egy adatbázis-kezelő rendszerrel kiegészíteni.

37

Köszönöm a figyelmet!

38