Objektum-orientált tervezési minták

OOP tervezési minták

- Tipikus de különböző köntösben visszatérő problémák objektum-orientált modelljei
- Példák tervezési mintákra (design patterns):
  - Sablonmetódus
  - Gyártó metódus
  - Iterátor
  - Toldalék
  - Homogén összetétel
  - Stratégia
  - …

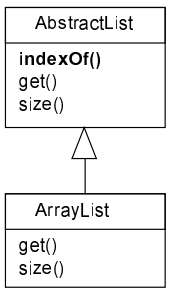
Tervezési minták leírása

- **Minta neve**: tömören és találóan leírja a minta koncepcionális tartalmát
- **Alapötlet**: probléma típusa, mi a megoldás célja
- **Esettanulmány**: alkalmazási mintapélda tervezésének leírása
- **Alkalmazhatóság**: a minta milyen helyzetekben alkalmazható, mozgástér, kompromisszumok, kötöttségek
- **Felelősségek és együttműködés**: tervezési minta osztályai, felelősségek hozzárendelése, együttműködés sémája

Sablonmetódus

- **Alapötlet**:
  - egy művelet megvalósításának vázát az ősosztályban absztrakt metódusok meghívásával definiáljuk
  - az absztrakt metódusokat a származtatott osztályban implementáljuk
- **Esettanulmány**
  - pl. az AbstractList osztály indexOf() művelete sablonmetódus
  - a get() és a size() absztrakt műveleteket a származtatott ArrayList osztályban implementáljuk

Sablonmetódus



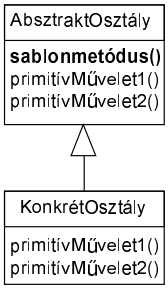
- **indexOf()**
  - definiálja egy adott elem megkeresésének algoritmusát
  - ehhez szükség van a get() és a size() metódusokra, melyek implementálását a származtatott ArrayList osztályra bízza

Alkalmazhatóság

- A sablonmetódust kódmegosztásra és elvonatkoztatásra használjuk
- A származtatott osztályok hasonló algoritmusainak vázát emeljük ki az ősosztályba
- A primitív műveletek implementálását a konkrét származtatott osztályokra hárítjuk

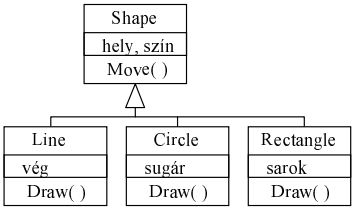
Felelősségek és együttműködés

- AbsztraktOsztály
  - implementálja az algoritmus vázát mint sablonmetódust
  - deklarálja az absztrakt elemi műveleteket
- KonkrétOsztály
  - az algoritmus vázát az AbsztraktOsztálytól örökli
  - megadja az elemi műveletek implementációját



Alkalmazási példa

- Move(): sablonmetódus, absztrakt draw() metódusokat hív meg
- Draw(): primitív művelet, melyet a származtatott osztályok implementálnak



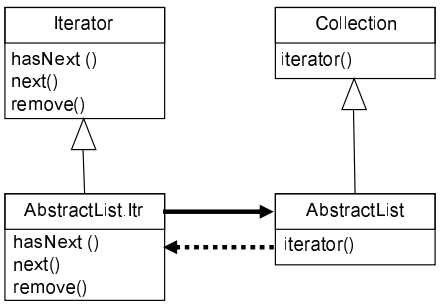
Gyártó metódus

- **Alapötlet**
  - a példányosítást egy absztrakt metóduson keresztül végezzük
  - a példányosítandó osztály kiválasztását nem kell a példányosítást igénylő kódra bízni

Gyártó metódus

- **Esettanulmány**
  - minden Collection interfészt implementáló adatszerkezet bejárásához szükség van egy Iterator interfészt implementáló objektumra
  - az iterator objektumot a Collection.iterator() **gyártó metódus** hozza létre
  - az iterator objektum konkrét típusát nem kell ismernünk a példányosításhoz
  - a Collection.iterator() implementációja ugyanakkor az adatszerkezetet reprezentáló osztály felelősége

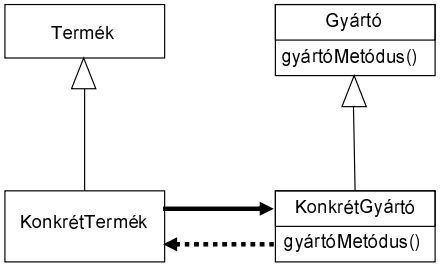
Esettanulmány



Alkalmazhatóság

- A példányosítást kezdeményező osztály nem akarja, vagy nem képes a példányosítandó osztályt rögzíteni
- Függetleníteni lehet egy interfész klienseit az interfész implementációjának kiválasztásától
- Gyakran párhuzamos típushierarchiákat köt össze

Felelősségek és együttműködés



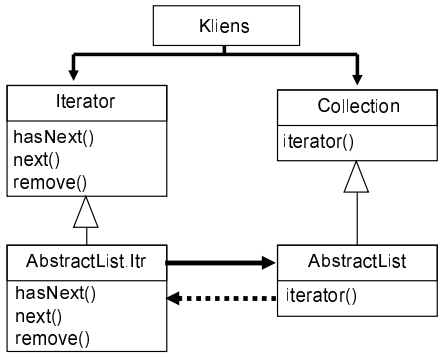
Felelősségek és együttműködés

- Termék
  - definiálja a gyártó metódus által példányosított osztályok őstípusát
- KonkrétTermék
  - implementálja a Termék interfészt
- Gyártó
  - deklarálja a gyártó metódust, melynek visszatérési értéke a Termék interfész implementációja
- KonkrétGyártó
  - implementálja a gyártó metódust

Iterátor

- Alapötlet
  - adatszerkezettől és absztrakt adattípustól független interfész aggregátumok bejárásához
- Esettanulmány
  - az adatszerkezet bejárásának mechanizmusát egy olyan független iterátor osztályba emeljük ki, amely implementál egy közös Iterator interfészt

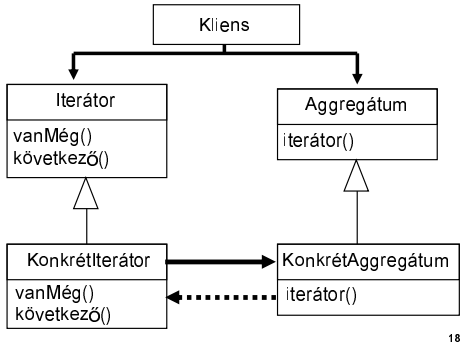
Esettanulmány



Alkalmazhatóság

- A bejáráshoz szükséges interfészt és implementációját függetleníthetjük az adatszerkezettől
- Közös őssel nem rendelkező aggregátumokat is egyazon interfésszel tudunk bejárni
- Az iterátor belső állapota független az aggregátumtól, így egyszerre több bejárás is folyamatban lehet
- Egy aggregátum többféle iterációt is támogathat (pl. lista esetén elölről hátra vagy hátulról előre)

Felelősségek és együttműködés



Felelősségek és együttműködés

- Iterátor
  - definiálja az adatszerkezet bejárásához szükséges interfészt
- KonkrétIterátor
  - implementálja az Iterátor interfészt
  - tárolja az iteráció aktuális állapotát
- Aggregátum
  - deklarálja az iterátort gyártó absztrakt metódust
- KonkrétAggregátum
  - implementálja a gyártó metódust, amely létrehozza a KonkrétIterátor osztály egy példányát

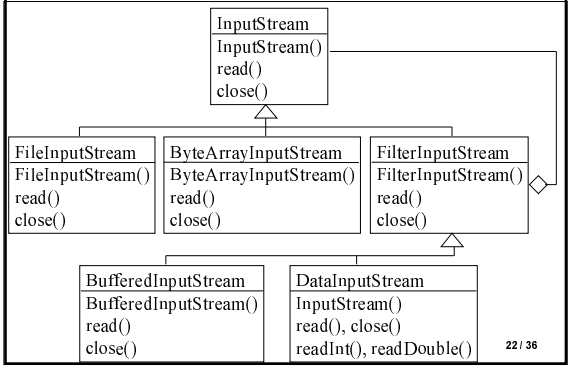
Toldalék

- Alapötlet
  - a funkcionális kiterjesztést nem öröklléssel, hanem objektumkompozícióval és delegációval valósítjuk meg
  - a kiterjesztések egymással dinamikusan kombinálhatók

Toldalék

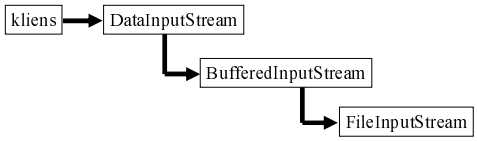
- Esettanulmány
  - input/output kezelése Jávában csatornákon keresztül történik
  - csatorna felelősségei:
    - fizikai tárolási szerkezet (pl. fájl vagy memória) hozzárendelése
    - pufferelés biztosítása
    - elemi típusok kódolása/dekódolása
  - a csatorna különböző felelősségei külön osztályokhoz vannak hozzárendelve
  - a csatorna bizonyos műveleteket nem maga hajt végre, hanem azokat egy alárendelt csatornához delegálja

Esettanulmány



Esettanulmány

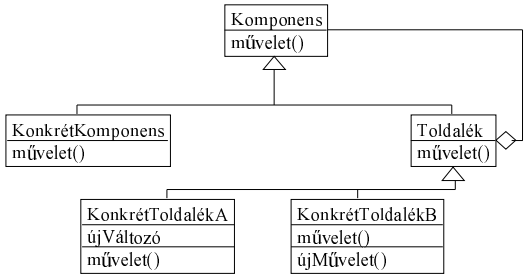
```
DataInputStream inputStream = new DataInputStream(
   new BufferedInputStream(new FileInputStream("file.dat")));
```



Alkalmazhatóság

- Egy osztály többféle, egymástól független kiterjesztésére van szükség, amelyeket még kombinálni is szeretnénk
- A toldalékos megoldás előnyei
  - toldalék több osztály példányaihoz is kapcsolható ⇒ kvázi többszörös öröklődés
  - toldalékok kombinálhatók
  - a toldalékot dinamikusan kapcsoljuk az objektumhoz
  - a toldalékok interfészen keresztül kapcsolódnak a kiterjesztett osztályhoz ⇒ függetlenül módosíthatók

Felelősségek és együttműködés



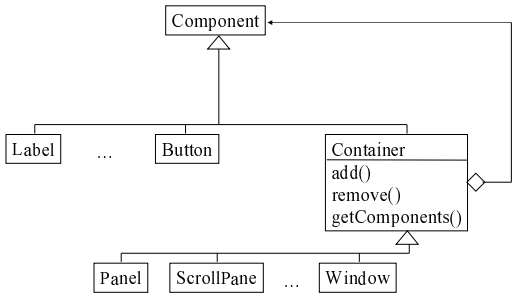
Felelősségek és együttműködés

- Komponens
  - definiálja a toldalékkal bővíthető objektumok közös interfészét
- KonkrétKomponens
  - definiál egy konkrét osztályt, melynek viselkedése toldalékokkal kiegészíthető
- Toldalék
  - egy komponens objektumhoz kapcsolódik, amelyhez delegálja a műveleteket
  - maga is kompatibilis a Komponens interfésszel
- KonkrétToldalék
  - új viselkedéssel terjeszti ki a Komponens interfész implementációit

Homogén összetétel

- Alapötlet
  - homogén rész-egész hierarchia elemeit egységes interfészen keresztül kezeljük
  - elemi és összetett objektumokat összefonó polimorfizmus
- Esettanulmány
  - AWT (Abstract Windowing Toolkit): a felhasználói felület komponenseit objektumok reprezentálják
  - elemi objektumok: címke (Label), gomb (Button)
  - összetett objektumok: ablak (Window), panel (Panel), görgetőablak (ScrollPane)
  - a hierarchikus építkezés végett az elemi és az összetett objektumokat egy egységes Component interfészen keresztül látjuk

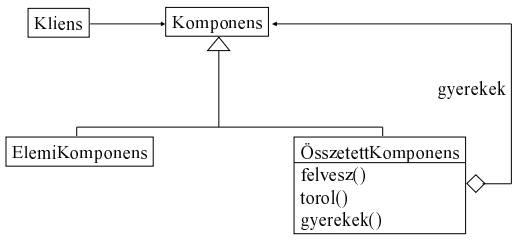
Esettanulmány



Alkalmazhatóság

- azonos őstípusú komponensekből álló rész-egész hierarchiák felépítése
- elemi komponensek polimorfizmusát kiterjesztjük az összetett komponensekre
- összetett komponensek rekurzív kompozíciója
- az összetett komponens kliensének nem kell foglakozni a gyerekek kezelésével

Felelősségek és együttműködés



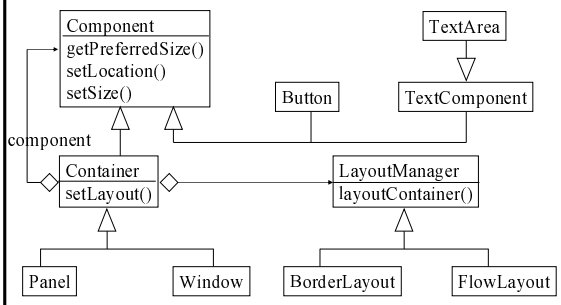
Felelősségek és együttműködés

- Komponens
  - definiálja a rész-egész hierarchia elemeinek közös interfészét
- ElemiKomponens
  - definiál egy elemi komponenst reprezentáló osztályt
- ÖsszetettKomponens
  - definiálja a gyerekek kezeléséhez szükséges interfészt és annak általános implementációját
  - a műveleteket gyakran rekurzívan továbbterjeszti
- Kliens
  - a rész-egész hierarchia elemeit a Komponens interfészen keresztül kezeli

Stratégia

- Alapötlet
  - egy algoritmust külön objektumként implementálunk
  - objektumkompozícióval választhatunk a variánsok közül
- Esettanulmány
  - AWT komponensek elrendezése sémák szerint történik
  - az elrendezés algoritmusa nem a Container leszármazottaihoz van hozzárendelve (redundancia és kombinatorikus robbanás kiküszöbölése)
  - az elrendezési sémák egy LayoutManager ősinterfészen keresztül külön hierarchiával vannak definiálva
  - az elrendezési séma a konténer stratégiája

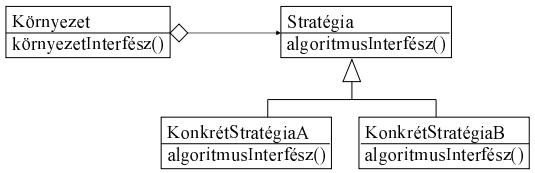
Esettanulmány



Alkalmazhatóság

- Az objektumok viselkedésének egy bizonyos aspektusát függetlenül akarjuk kezelni:
  - nem csak egy osztály, hanem egy osztályhierarchia viselkedését akarjuk variálni (kombinatorikus robbanás elkerülése)
  - egy adott viselkedést leíró kód egységbe zárása
  - az eredeti osztály egyszerűsítése azáltal, hogy a viselkedés egy aspektusa jól elkülönül egy interfészen keresztül

Felelősségek és együttműködés



Felelősségek és együttműködés

- Stratégia
  - definiálja a viselkedési variációk közös interfészét
- KonkrétStratégia
  - megvalósít egy konkrét viselkedési variációt
- Környezet
  - viselkedésének egy részét a KonkrétStratégia objektummal együttműködve határozza meg
  - a kommunikáció a Stratégia interfészen keresztül történik
  - átadja a variáció megvalósításához szükséges adatokat, vagy a saját referenciáját a KonkrétStratégia objektumnak