8. Strukturális tervezési minták I.

Adapter (Structural pattern)

- Egy osztály interfészét olyan interfésszé konvertálja, amit a kliens vár.
- Lehetővé teszi olyan osztályok együttműködését, amik egyébként inkompatibilis interfészeik miatt nem tudnának együttműködni.
- **Probléma**
  - Össze szeretnénk kötni két rendszert, amik nem kompatibilisek.
    - Például van egy alkalmazás, ami XML formátummal működik és szeretnénk használni egy másik csomagot, ami csak JSON formátummal működik.
- **Megoldás**
  - *(Valós példa: Adapter kábelek: VGA -> HDMI és vissza)*
  - Készítünk egy adapter-t, ami elrejti magát a konverziót.

Adapter használjuk, ha

- Egy olyan osztályt szeretnénk használni, aminek interfésze nem megfelelő Adapter.
- Egy újrafelhasználható osztályt szeretnénk készíteni, ami együttműködik előre nem látható vagy független szerkezetű osztályokkal. (pluggable adapters)

Adapter implementálása

1. Adapter osztály elkészítése
2. Az adapter osztályban adjunk hozzá egy field-et, ami referenciaként rámutat a service objektumra.
3. Kliens interfész metódusainak implementálása az adapter osztályban.
4. Hajtsuk végre magát a konverziót az adapter segítségével a két nem kompatibilis interfész között.

Adapter előnyök és hátrányok

- **Előnyök**
  - Single Responsibility elv
  - Open/Closed elv
- **Hátrányok**
  - Komplexitás növekedhet minden egyes új osztálynál és interfésznél.

Bridge (Structural pattern)

- Különválasztja az absztrakciót (interfészt) az implementációtól, hogy egymástól függetlenül lehessen őket változtatni.
- **Probléma**
  - Egy osztály két jellemzőtől is függ
  - Például alakzatok, szín és forma
- **Megoldás**
  - Szét kell bontani az osztályt
  - A forma osztály várja interfészen keresztül a szín osztályt
  - Kompozícióval lehessen összeépíteni őket

Bridge használjuk, ha

- Egy osztályt több ortogonális (független) dimenzióban kell bővíteni.
- Futás közben implementációt szeretnénk váltani

Bridge implementálása

1. Bridge interfész létrehozása.
2. Bridge osztály létrehozása, ami implementálja a Bridge interfészt.
3. Abstract osztály létrehozása
4. Konkrét osztály létrehozása, ami implementálja az Abstract osztályt

Bridge előnyei és hátrányai

- **Előnyök**
  - Absztrakció és az implementáció különválasztása
  - Az implementáció dinamikusan, akár futási időben is megváltoztatható
  - Az implementációs részletek a klienstől teljesen elrejthetők
  - Az implementációs hierarchia külön lefordított komponensbe tehető, így ha ez ritkán változik, nagy projekteknél nagymértékben gyorsítható a fordítás ideje
  - Ugyanaz az implementációs objektum több helyen is felhasználható
- **Hátrányok**
  - Bonyolulttá válhat a kód egy idő után

Composite (Structural pattern)

- Másnéven **Object Tree**
- **Probléma**
  - Nehezen tudunk az objektumainkból hierarchikus rendszert építeni.
  - Például részlegek és dolgozók korrekt ábrázolása.
  - Egy részfa vagy akár egy levélelem is ugyanazt a szolgáltatáskészletet nyújtsa.
- **Megoldás**
  - Fa szerkezet építése
  - Egy csomópontnak tetszőleges mennyiségű gyermekeleme legyen.
  - A csomópontnak és levél elemek is ugyanazt az interfészt valósítsák meg.
  - Lehessen rekurzívan bejárni.

Composite implementálása

1. Alkalmazás alapvető modellje fa struktúraként ábrázolható kell legyen.
2. Komponens interfész implementálása
3. Levélosztály létrehozása az egyszerű elemek ábrázolására.
4. Osztály létrehozása az összetett elemek ábrázolásához.
    a. Tömböt létre kell hozni, amiben az alelemekre való hivatkozásokat tárolja.
    b. Tömbnek képesnek kell lennie a levelek, konténerek tárolására is, ezért a komponens interfész típusával kell deklarálni.
5. Metódusok deklarálása, amivel hozzáadhatunk vagy törölhetünk gyermekelemeket.

Composite használjuk, ha

- Objektumok rész-egész viszonyát szeretnénk kezelni.
- A kliensek számára el akarjuk rejteni, hogy egy objektum egyedi objektum vagy kompozit objektum.
    - Bizonyos szempontból egységesen szeretnénk kezelni őket.

Composite előnyök és hátrányok

- **Előnyök**
    - Összetetteb fa struktúrával is dolgozhatunk.
    - Open/Closed elv
- **Hátrányok**
    - Nehéz lehet közös interfészt biztosítani, mivel a funkcionalitások eltérhetnek.

Flyweight (Structural pattern) trükkök

- Nincs konkrét megoldás, sok trükköt biztosít a Flyweight minta.

1. On-the-fly property-k

- A memóriában nem foglalnak helyet ezek a property-k.
- Amikor az adott property-t lekérjük, akkor lazy loading elven akkor hajtódik végre, amikor szükség van rá.
- Amikor a főprogram elkéri az adott property-t, akkor hajtódik végre a „levegőben”, emiatt nevezzük on-the-fly property-nek.
- El kell dönteni, hogy mikor akarjuk használni, mert például ha rengeteg adat van és például azokon akarunk átlagolni, akkor az sokáig is eltarthat.
- Ha nem használjuk, akkor pedig használjunk külön szálakat, aszinkron metódusokat például.

2. Objektumok közös részeinek eltárolása egyszer

- Példány szintjén is megnézhetjük az adott tulajdonságot.
- Felesleges tárolást lehet vele kiváltani, mert olyan jellemzőket teszünk bele, amiket nem szeretnénk módosítani.
- Mivel ez egy megosztott objektum és ha átírunk valamit, akkor az összes többi példányra kihatással van.
- Így érdemes védeni az írás ellen, tehát olvashatóként kell definiálni.

3. Újrahasznosított objektumok

- Lényege, hogy ne hozzunk létre újabb objektumot például egy törlés után, hanem használjuk fel újra a már meglévőt.
- Memóriát és CPU időt is megtakaríthatunk vele, mert mindig ugyanazt az objektumot használjuk fel.

Flyweight a .NET osztályokban (String, Type)

- **String**-ek .NET-ben immutable-ek, vagyis nem lehet létrehozás után módosítani.
    - Gyorsítótárba helyezi újrafelhasználás céljából.
    - Tehát megnézi, hogy van-e már egy ugyanilyen értékű létező String a String pool-ban, ha van, akkor nem jön létre új String, hanem a meglévő String-re való hivatkozás kerül vissza.
- A **Type** osztály egy objektum típusát reprezentálja és minden típusnak egyedi identitása van egy AppDomain-en belül.
    - Típusokat metaadatokból tölti be és a típus metaadatai az újrafelhasználás miatt gyorsítótárba kerülnek.
    - Ezek a metaadatok tartalmazzák a típus nevére, névterére, attribútuamira és member-ekre vonatkozó információkat.
    - Tehát a gyorsítótárazott metaadatokat adja vissza, ahelyett, hogy a metaadatokat újratöltené a lemeről.