Szoftvertechnológia

ÓBUDAI EGYETEM
NEUMANN JÁNOS INFORMATIKAI KAR

MODUL 8

Viselkedési tervezési minták

Iterator

Chain of responsibility

Visitor

Command

Observer

Mediator

GOF minták

23 db minta		Cél		
		Létrehozási/Creational	Strukturális/Structural	Viselkedési/Behavioral
Hatókör	Osztály	Factory method	Adapter	Interpreter Template method
	Objektum	Abstract factory Builder Prototype Singleton	Adapter Bridge Composite Decorator Facade Flyweight Proxy	Chain of responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

Iterator

- **Probléma**: Legyen bármilyen gyűjteményünk (lista, fa, gráf, hasítótábla, stb.) ezeket szeretnénk egy bejárható interfészen keresztül elérni
- **Megjegyzés**
  - Korábban már tanultunk róla a láncolt listánál
  - LINQ is erre épít
  - **IEnumerable<T>** interfészen át érjük el a gyűjteményeinket
- **Megoldás**
  - Az adatszerkezetünkön implementáljuk az **IEnumerable<T>** és egy külső bejáró osztályon az **IEnumerator<T>** interfészt
  - Az **IEnumerator<T>** előírja az alábbi metódusokat:
    - **void Reset()** → gyűjtemény elejére visszaállás
    - **bool MoveNext()** → következő elemre lépés
    - **T Current** → visszaadja az aktuális elemet
  - Vagy **yield return** → lásd példa

Chain of responsibility I.

- **Probléma**
  - Egy olyan folyamatot implementálunk, ahol egy objektum sok részfeladaton megy keresztül
    - Pl: 10 db validációt egymás után meghívunk rajta → ha bármelyik hibára fut, nem nézzük a többit (feltételes továbbpasszolás)
    - Pl: 10 db naplózó metódusnak egymás után átadjuk az objektumot → mindenképp mindegyiken végigmegy (feltétel nélküli továbbpasszolás)
  - Első esetben egy nagyon összetett elágazás jön létre
  - Utóbbi esetben pedig 10 db metódushívás egymás alatt
- **Megoldás**
  - Készítünk a validáló/logoló osztályokból egy gyűjteményt
  - Végigmegyünk az elemein és mindegyiknek átadjuk az objektumot

Chain of responsibility II.

- **Probléma**
  - Egy olyan folyamatot implementálunk, ahol egy fizetési objektum egy döntési hierarchián fut végig
    - Osztályvezető → 10.000 ft alatt hagyhatja jóvá
    - Középvezető → 100.000 ft alatt hagyhatja jóvá
    - Felsővezető → Minden más esetben hagyhatja jóvá
  - Csak a közvetlen feletteshez intézünk kérelmet, láthatatlan számunkra, hogy ő maga saját hatáskörben jóváhagyta, vagy pedig a saját feletteséhez fordult → csak kapunk egy választ, hogy igen/nem
- **Megoldás**
  - Elkészítünk egy Boss osztályt
  - A példányok tárolják a saját pénzügyi hatókörüket
  - A példányok tárolják a saját felettesüket
  - A példányok döntést hoznak vagy továbbítják a kérelmet felfele

Visitor

- **Probléma**
  - Szeretnénk a **hívó**t és a **hívott**at szétválasztani
  - A **hívó** (**Logic**) tudhat a **hívott**ról (**Subject**), de fordítva tilos
  - A hívott dönthessen róla, hogy vele éppen lehet-e dolgozni
  - Működjön az egész gyűjteményekkel is (több **hívó** és több **hívott**)
- **Megoldás**
  - Interfészeken át érjék el egymást
  - **Hívott**nak legyen: **Accept()** metódusa
  - **Hívó**nak legyen: **Visit()** metódusa
  - A **hívott** az **Accept()** metódusában döntést hoz és egyben meghívja a **hívó Visit()** metódusát

Observer

- **Probléma**
  - A Subject állapotváltozásairól szeretnénk különböző feliratkozókat értesíteni
  - De nem igazán kéne tudnunk róla, hogy milyen feliratkozók vannak és hogy vannak-e
  - Pl: változott egy személy fizetése → minden felületen, adattárolóban módosítsuk
- **Megoldás**
  - Feliratkozó osztályok valósítsanak meg egy **ISubscriber** interfészt
  - Ez írjon elő egy **StateChange()** metódust
  - A **subject** kezelje a feliratkozókat (**Subscribe()**, **UnSubscribe()**)
  - Állapotváltozáskor hívja meg az összes feliratkozó **StateChange()** metódusát
  - A feliratkozók tegyék meg a frissítési lépéseket

Command

- **Probléma**
  - Egyirányú függés van két nagy réteg között
  - **UI** ismeri a **Logic**-ot (példában **Repository**-t)
  - Ha lecseréljük egy másik **Repository**-ra, akkor nagyon sok helyen a **UI** kódját át kell írnunk
- **Megoldás**
  - Közvetve ismerje csak a **UI** a **Repository**-t
  - A közvetítő osztályban kelljen átírni bármit, ha módosul a **Repository**
  - A közvetítő osztályba pedig extra kód is legyen írható

Mediator

- **Probléma**
  - Egyirányú függés van két nagy réteg között
  - Ne legyen semmilyen irányú függés
  - Egy közvetítő osztályon keresztül lehessen csak beszélgetni két osztálynak
- **Megoldás**
  - Legyen egy **Mediator** osztályunk
  - Lehessen regisztrálni egy üzenetfolyamára
  - Lehessen beküldeni egy üzenetet valamelyik üzenetfolyamára

Köszönöm a figyelmet!

Kérdés esetén e-mailben szívesen állok rendelkezésre.