**Tervezési minták egy OO programozási nyelvben.**

**MVC,mint modell-nézet-vezérlő minta és néhány másik tervezési minta.**

Szilágyi Mihály Dávid

Q6P7LD

Tartalomjegyzék

[Tervezési minták 4](#_Toc152734550)

[Tervezési minták elemei 4](#_Toc152734551)

[Tervezési minták csoportosítása 5](#_Toc152734552)

[A gyártási minták 5](#_Toc152734553)

[Factory Method 5](#_Toc152734554)

[Abstract Factory 5](#_Toc152734555)

[Builder 5](#_Toc152734556)

[Prototype 5](#_Toc152734557)

[Singleton 5](#_Toc152734558)

[A szerkezeti minták 6](#_Toc152734559)

[Adapter 6](#_Toc152734560)

[Bridge 6](#_Toc152734561)

[Composite 6](#_Toc152734562)

[Decorator 6](#_Toc152734563)

[Facade 6](#_Toc152734564)

[Flyweight 6](#_Toc152734565)

[Proxy 6](#_Toc152734566)

[A viselkedési minták 7](#_Toc152734567)

[Interpreter 7](#_Toc152734568)

[Template Method 7](#_Toc152734569)

[Chain of Responsibility 7](#_Toc152734570)

[Command 7](#_Toc152734571)

[Iterator 7](#_Toc152734572)

[Mediator 7](#_Toc152734573)

[Memento 7](#_Toc152734574)

[Observer 7](#_Toc152734575)

[State 7](#_Toc152734576)

[Strategy 8](#_Toc152734577)

[Visitor 8](#_Toc152734578)

[MVC, mint modell-nézet-vezérlő minta 9](#_Toc152734579)

[A modell 9](#_Toc152734580)

[A nézet 9](#_Toc152734581)

[A vezérlő 9](#_Toc152734582)

Tervezési minták

A tervezési minták sokszor felmerülő problémák leírásait adják meg úgy, hogy közben megadják a megoldás magját, amit adaptálva a konkrét rendszerhez újra és újra alkalmazhatunk. Ami a kódolásnak az osztály könyvtár, az a tervezésnél a tervezési minták gyűjteménye.

Objektum orientált programok esetében a tervezési minta leírása megadja azokat az egymással kommunikáló objektumokat, osztályokat, amelyek együttes viselkedése az adott problémára megoldás lehet.

A legtöbb jó OO architektúra használ mintákat, mivel a minták által a rendszer egyszerűbb, karbantarthatóbb és újrafelhasználhatóbb lesz, valamint a minták ismeretével megérteni is könnyebb a rendszert.

# Tervezési minták elemei

A tervezési minták megadásakor meg kell adjunk annak nevét. Ez fontosabb elem, mint ahogy azt gondolnánk, sokszor a név alapján már beazonosítható az, hogy a minta mely problémára is próbál megoldást nyújtani.

Mindenképpen tartalmaznia kell a minta leírásának magát a problémát. Tudni kell, hogy az adott minta milyen helyzetben, milyen kontextusban használható.

A mintának meg kell adnia a megoldást is az adott problémára. Ez a minta elemeinek absztrakt leírásából, azok között levő kapcsolatokból, feladatokból fog állni. Konkrét implementációt nyilván nem ad a minta, azt mindig adaptálni kell az aktuális rendszerhez.

Fontos, hogy a minták használatának lehetnek pozitív és negatív következményei, általában ezeket is célszerű megadni a mintáknál. Hátrány lehet, ha a minta alkalmazása rontja a program futásának idejét, ugyanakkor sokszor a kód sokkal áttekinthetőbb és újrafelhasználhatóbb lesz általa.

# Tervezési minták csoportosítása

A mintákat legfőbb tulajdonságaik szerint csoportokba rendezhetjük. Ez a csoportosítási szempont lehet a rendeltetésük, vagy a hatáskörük.

Rendeltetés szerint lehetnek gyártási mintáink, amelyek alapvetően az objektumoknak a létrehozási módjáért felelősek, lehetnek szerkezetik, amik meghatározzák objektumok vagy osztályok felépítését, illetve lehetnek viselkedési minták, amik osztályok és objektumok közötti kölcsönhatásokat írják le, illetve azt, hogyan osztják meg egymás között a felelősségeket.

Hatáskör szerint csoportosítva a mintákat megkülönböztetünk olyanokat, amik osztályok közötti statikus, vagy amik objektumok között levő dinamikus kapcsolatokat írnak le.

## A gyártási minták

A gyártási minták a példányosítás folyamatában segítenek oly módon, hogy maga a folyamat elkülönüljön a példányosítás módjától. Az hogy mi jön létre, hogyan jön létre, ki hozza létre az nem fontos, csak a létrehozott objektum.

Factory Method (Virtual Constructor) - Gyártófüggvény: Interfészt definiál egy objektum létrehozásához, de az alosztályokra bízza, melyik osztályt példányosítják. A factory method megengedi az osztályoknak, hogy a példányosítást az alosztályokra ruházzák át.

Abstract Factory (Kit) - Elvont gyár: Kapcsolódó vagy egymástól függő objektumok családjának létrehozására szolgáló interfészt biztosít a konkrét osztályok megadása nélkül.

Builder - Építő: Összetett objektum létrehozását biztosítja az elemeinek létrehozásával és összerakásával.

Prototype - Prototípus: Prototípus példány használatával egy alap objektumot, az újabb objektumok létrehozását pedig ennek a prototípusnak a lemásolásával állítja elő.

Singleton - Egyke: Egy osztályból csak egy példányt engedélyez, és ehhez globális hozzáférést ad.

## A szerkezeti minták

A szerkezeti mintákat szintén minden területén használhatjuk a szoftverfejlesztésnek. Céljuk az, hogy objektumok, vagy osztályok összetételével nagyobb szerkezeteket hozzunk létre, amelyek kezelése így áttekinthetőbb.

Adapter (Wrapper) - Illesztő: Az adott osztály interfészét átalakítja, hogy az kompatibilissé (használhatóvá) váljon más rendszerek részére.

Bridge (Handle, Body) - Híd: Az elvont ábrázolás (absztrakció) és implementáció közötti csatolást lazítja, hogy a kettő egymástól függetlenül legyen változtatható.

Composite - Összetétel: Rész-egész szerkezetek leképezését valósítja meg az objektum-hierarchiára a rész és egész azonos kezelésével.

Decorator (Wrapper) - Díszítő: További felelősségeket/tulajdonságokat csatol dinamikusan az objektumhoz.

Facade - Homlokzat: Egy alrendszerben különböző interfészek egy halmazához egységes interfészt biztosít. A módszerrel magasabb szintű interfészt határozunk meg, amelynek révén az adott alrendszer könnyebben használhatóvá válik.

Flyweight - Pehelysúlyú: Nagy számú apró, megosztott objektum kezelését teszi hatékonnyá. Akkor használatos, ha nagy (nagyságrendileg száz, vagy ezer) számú objektumunk van, amelyek nagyon hasonlóak. Ilyenkor ez a minta csökkentheti a memória felhasználást.

Proxy (Surrogate) - Helyettes: Egy adott objektumot egy helyettesítő objektummal vált fel, amely szabályozza az eredeti objektumhoz történő hozzáférést is. Gyakorlatilag a proxyval egy köztes réteg kerül a kliens és az objektum közé. Így az eredeti objektumhoz való hozzáférést kontrollálhatjuk.

## A viselkedési minták

A viselkedési minták elsősorban algoritmusokkal, illetve az osztályok és objektumok közötti kommunikációval, a felelősségi körök kijelölésével foglalkoznak. Az osztály viselkedési minták általában öröklődéssel valósulnak meg, míg az objektum viselkedési mintáknál egyenrangú objektumok kommunikálnak anélkül, hogy mindegyik ismerje mindegyiket.

Interpreter - Értelmező: Egy adott nyelv nyelvtanát reprezentálja megadva azt az értelmezőt is, amely értelmezni tudja az adott nyelven leírt mondatokat.

Template Method - Sablonfüggvény: Egy algoritmus vázát definiálja, amelyben bizonyos műveleteket csak absztrakt módon definiál, megvalósításukért csak a származtatott osztályok lesznek felelősek.

Chain of Responsibility - Felelősséglánc: A kérés/parancs küldője és végrehajtója közötti közvetlen csatolást megszünteti úgy, hogy a kérést feltételezhetően kezelni tudó objektumokat láncba állítja, és a kérés addig halad a kiszolgálók láncolatán, amíg azt valamelyik le nem kezeli.

Command (Action, Transaction) - Parancs: Kéréseket objektumként ábrázolja azok sorba állíthatósága, naplózhatósága végett.

Iterator (Cursor) - Iterator: Tároló-objektum elemeinek sorozatos elérését teszi lehetővé a reprezentációtól függetlenül.

Mediator - Közvetítő: Objektumok együttműködési módját változtatja meg egy egységbezáró objektum közbeékelésével, amely által a két objektum közötti eredeti csatolás megszűnik.

Memento (Token) - Emlékeztető: Az egységbe zárás megsértése nélkül kinyeri és elmenti egy objektum belső állapotát, hogy az később ebbe az állapotba visszaállítható legyen.

Observer (Dependents, Publish-Subscribe) - Megfigyelő: Egy objektum állapotát megfigyeli, az objektum állapotának megváltozásáról pedig a tőle függő objektumokat értesíti.

State (Objects for States) - Állapot: Adott objektum számára engedélyezi, hogy belső állapotának megváltozásával megváltoztathassa viselkedését is. Az objektum ekkor látszólag módosítja az osztályát.

Strategy (Policy) - Stratégia: Meghatároz egy algoritmus családot, amelyben a több, azonos feladatú, de különböző algoritmus egységbezárásával azok felcserélhetővé válnak.

Visitor - Látogató: Egy adott objektum-hierarchián végezhető műveletsort reprezentál, a hierarchia bejárásával ezen műveletek elvégezhetőekké válnak a hierarchia elemein annélkül, hogy a a hierarchiát alkotó osztályokat megváltoztatnánk.

# MVC, mint modell-nézet-vezérlő minta

A model-view-controller (modell-nézet-vezérlő), röviden MVC struktúra az 1970-es években lett kifejlesztve, amikor a grafikai kezelőfelületek fejlesztésük korai szakaszában voltak még. Az MVC célja elválasztani az üzleti logikát a felhasználói felülettel kapcsolatos elgondolásoktól oly módon, hogy a fejlesztők minél könnyebben változtathassák meg az egyes részeket anélkül, hogy a többire hatással lennének. Az MVC arra törekszik, hogy egy alkalmazást három részre bontson:

## A modell

Itt tároljuk az adatokat és az üzleti logikát. Egy tipikus webapplikációban ez a rész foglalkozik például az adatbázisokkal és egyéb objektumokkal amivel az applikáció dolgozik.

## A nézet

A nézet a tényleges vizuális leképzése a modellednek. Egy tipikus webalkalmazásban ez lenne például a weblap ami megjeleníti a modellt a felhasználó részére, legyen az egy adatbevitelre használt űrlap, adatkivitelre használt mezők, vagy a kettő keveréke. Természetesen a nézetnek nem kell egy modell minden részét mutatnia, és egy modellhez tartozhat több nézet is.

## A vezérlő

A vezérlő kezeli a modell és a nézet közti kommunikációt. Egy webalkalmazásban a vezérlő metódusai vannak végrehajtva amikor a felhasználó egy weblapot tölt be vagy egy gombra kattint. A vezérlő ezekkel frissíti a modellt, ha szükséges, majd új nézetet ad vissza, ha pedig az szükséges.