**Tervezési minták egy OO programozási nyelvben,**

**MVC mint modell nézet vezérlő minta és néhány másik tervezési minta**

1. **Tervezési minták egy OO programozási nyelvben:**

A tervezési minták (design patterns) olyan ismétlődő megoldások, amelyeket a szoftverfejlesztés során gyakran felmerülő problémák kezelésére dolgoztak ki. Ezek a minták nem konkrét kódot jelentenek, hanem magas szintű megoldásokat, amelyeket különböző programozási nyelvekben megvalósíthatunk, beleértve az objektumorientált (OO) nyelveket, mint a Java, C#, vagy C++.

A tervezési minták jelentős szerepet játszanak az objektumorientált programozásban, mivel segítenek a gyakori tervezési problémák megoldásában, miközben biztosítják a kód karbantarthatóságát és rugalmasságát. Bár a minták alkalmazása nem kötelező, azok ismerete és helyes használata hozzájárul a hatékonyabb és rendezettebb szoftverfejlesztéshez.

* **A tervezési minták alapjai:**

Az OO programozásban a szoftverek építőelemei az osztályok és az objektumok. Az objektumok állapotot tárolnak és viselkedéseket definiálnak. Az OO tervezési minták célja, hogy az objektumok közötti kapcsolatokat és interakciókat hatékonyan szervezzék meg, ami hozzájárul a rugalmasabb, bővíthetőbb és karbantarthatóbb rendszerek létrehozásához.

* **A tervezési minták típusai:**

A tervezési mintákat általában három kategóriába sorolják:

* + Kreációs (creational) minták: Az objektumok létrehozását optimalizálják, hogy a rendszer rugalmas maradjon.

*Példa: Singleton minta: Gondoskodik arról, hogy egy osztálynak csak egyetlen példánya létezzen.*

* + Strukturális (structural) minták: Ezek a minták az objektumok és osztályok közötti kapcsolatokat és összekapcsolási módokat segítenek optimalizálni.

*Példa: Adapter minta: Lehetővé teszi, hogy két, egymással nem kompatibilis interfészt használó osztály együttműködjön.*

* + Viselkedési (behavioral) minták: A rendszeren belüli objektumok közötti kommunikáció módjára fókuszálnak.

*Példa: Observer minta: Lehetővé teszi, hogy egy objektum (az alany) értesítse a hozzá kapcsolódó megfigyelőket, amikor az állapota megváltozik.*

Minták:

* + Kreációs minták:
* Factory Method:

Ahogy már említettem, a Factory Method biztosít egy interfészt az objektumok létrehozásához, de magát a konkrét példányosítást a leszármazott osztályokra bízza.

*Példa: Egy játékban a különböző szörnyek létrehozására használható, ahol a konkrét szörny típusa csak a futásidőben derül ki.*

* Abstract Factory:

Egy olyan minta, amely egy interfészt definiál a kapcsolódó vagy egymástól függő objektumok családjainak létrehozásához, anélkül, hogy azok konkrét osztályait meghatároznánk. Ez egyfajta "gyárak gyára", hiszen több, egymással kapcsolódó Factory Methodot egyesít.

*Példa: Egy grafikus felület létrehozásakor különböző platformokra (Windows, macOS) készíthetünk különböző gombokat és ablakokat, anélkül, hogy konkrét osztályokra hivatkoznánk.*

* Singleton:

Biztosítja, hogy egy osztálynak csak egyetlen példánya legyen, és egy globális hozzáférési pontot biztosít ehhez az objektumhoz.

*Példa: Egy adatbázis kapcsolat kezelése, amelynél csak egyetlen példányt szeretnénk használni a teljes alkalmazásban.*

* Builder:

Olyan minta, amely bonyolult objektumok fokozatos létrehozását teszi lehetővé, lépésről lépésre. A Builder minta célja az, hogy különválassza az objektumok létrehozásának logikáját a végeredmény struktúrájától.

*Példa: Egy házépítés során, ahol különböző részek (pl. falak, ajtók, tető) kerülnek hozzáadásra lépésenként.*

* Prototype:

A Prototype minta lehetővé teszi objektumok létrehozását egy meglévő objektum másolásával (klónozással), ahelyett, hogy új példányt hoznánk létre. Ez különösen akkor hasznos, ha a példányosítás költséges.

*Példa: Egy grafikus alkalmazásban, ahol egy meglévő elemet szeretnénk sokszorosítani az új objektumok előállításához.*

* Object Pool:

A minta célja az erőforrások hatékony kezelése azáltal, hogy egy adott erőforrást újra felhasználunk ahelyett, hogy új példányokat hoznánk létre, amikor szükség van rá. Egy objektum poolban objektumokat előre létrehozunk, majd újrafelhasználjuk őket.

*Példa: Adatbázis kapcsolat poolok, amelyek új kapcsolat létrehozása helyett újra felhasználják a meglévő kapcsolatokat.*

* + Strukturális minták:
* Adapter (más néven Wrapper minta):

Az Adapter minta célja, hogy két inkompatibilis interfészt összekapcsoljon, vagyis egy osztály interfészét átalakítja úgy, hogy egy másik osztály is használhassa azt.

*Példa: Ha van egy régi rendszer, amely más formátumú adatokat használ, az adapter biztosítja, hogy egy új rendszer kompatibilis legyen ezzel az interfésszel anélkül, hogy mindkét rendszert módosítani kellene.*

* Bridge:

A Bridge minta lehetővé teszi az absztrakció és annak implementációjának különválasztását, így függetlenül fejleszthetők tovább. Ez segít elkerülni a túl sok alosztály létrehozását.

*Példa: Egy rajzolási program, ahol a különböző formákat (kör, négyzet) külön lehet kezelni a megjelenítési módszerektől (pl. Vektoros, Raszteres).*

* Composite:

A Composite minta lehetővé teszi, hogy egyedi objektumokat és objektumcsoportokat ugyanúgy kezeljünk. Ezáltal egy hierarchikus szerkezetben bármely csomópont és levél ugyanolyan műveleteket támogathat.

*Példa: Egy fájlrendszer, ahol mappákat és fájlokat ugyanúgy kezelhetünk, például egy fájlméret lekérdezésekor egy mappa összes fájlméretét összeadjuk.*

* Decorator:

A Decorator minta lehetővé teszi, hogy objektumok funkcionalitását dinamikusan bővítsük anélkül, hogy új alosztályokat kellene létrehozni. A dekorátor új viselkedéseket ad az eredeti objektumhoz.

*Példa: Egy kávézói rendelés kezelése, ahol a kávé alapját dinamikusan kiegészíthetjük például tejjel, cukorral vagy habbal anélkül, hogy minden kombinációra új osztályt hoznánk létre.*

* Facade:

A Facade minta célja, hogy egy egyszerűsített interfészt biztosítson egy összetett alrendszerhez. Ezáltal elrejthetjük a komplexitást a felhasználók elől, és csak azokat a funkciókat tesszük elérhetővé, amelyekre valóban szükség van.

*Példa: Egy számítógép rendszer bekapcsolása, ahol a felhasználó csak a "bekapcsoló gombot" nyomja meg, de a háttérben sok összetett művelet (tápellátás, inicializálás stb.) történik.*

* 6. Flyweight:

A Flyweight minta célja, hogy minimalizálja az objektumok létrehozásának költségeit azáltal, hogy megosztja azokat a közös adatokat, amelyek nem változnak. Ezáltal csökkenthető a memóriahasználat és a rendszer terhelése.

*Példa: Egy szövegszerkesztőben, ahol az egyes karakterek vizuális megjelenítését (pl. betűtípus, stílus) közös objektumokkal lehet megosztani ahelyett, hogy minden egyes karakternek külön objektumot hoznánk létre.*

* Proxy:

A Proxy minta egy helyettes objektumot biztosít, amely közvetítőként működik az eredeti objektum elérése előtt. Ezáltal az eredeti objektum hozzáférése kontrollálható, például amikor az objektum erőforrásigényes vagy távoli helyen található.

*Példa: Egy távoli szerveren lévő adatbázis elérésekor egy Proxy objektumot használunk, amely csak akkor csatlakozik a szerverhez, amikor tényleg szükséges.*

* Private Class Data (ritkábban említik strukturális mintaként):

Ez a minta lehetővé teszi, hogy egy osztály belső adatait védjük úgy, hogy csak meghatározott metódusokon keresztül érhetők el, elkerülve a közvetlen adatmanipulációt.

*Példa: Egy banki alkalmazás, ahol a felhasználó adatai védve vannak, és csak bizonyos lekérdezési vagy módosítási metódusokon keresztül elérhetők.*

* + Viselkedési minták:
* Chain of Responsibility (Felelősség-lánc):

Ez a minta lehetővé teszi, hogy egy kérés több objektumon keresztülhaladjon addig, amíg valaki kezelni nem tudja. Minden objektumnak lehetősége van kezelni a kérést, vagy továbbadni azt a lánc következő elemének.

*Példa: Egy ügyfélszolgálati rendszer, ahol a kérés először az első szintű ügyfélszolgálathoz jut, és ha ők nem tudják kezelni, továbbítják a kérdést a következő szintre.*

* Command (Parancs):

A Command minta parancsokat objektumként kezel, lehetővé téve a parancsok végrehajtását, sorba állítását, vagy akár visszavonását. Ez a minta elválasztja az akció indítóját az akció végrehajtójától.

*Példa: Egy szövegszerkesztőben a másolás, beillesztés vagy visszavonás funkciók mind parancsokként kezelhetők, amelyek sorba állíthatók vagy visszavonhatók.*

* Interpreter (Értelmező):

Az Interpreter minta lehetővé teszi, hogy egy adott nyelv szintaxisát és szabályait definiáljuk, és ennek megfelelően értelmezzük a bemenetet. Ez a minta gyakran használatos nyelvértelmezők vagy konfigurációs fájlok feldolgozásához.

*Példa: Egy egyszerű matematikai kifejezés kiértékelése, ahol a kifejezést (pl. 5 + 3) egy értelmező objektum bontja le és dolgozza fel.*

* Iterator (Iterátor):

Az Iterator minta lehetővé teszi, hogy egy kollekció elemein egységes módon végighaladjuk anélkül, hogy feltárnánk a kollekció belső szerkezetét.

*Példa: Egy lista vagy tömb elemein végig lépés, például egy fájlrendszerben található fájlok listájának bejárása.*

* Mediator (Mediátor):

A Mediátor minta biztosít egy közvetítőt az objektumok közötti kommunikációhoz. A közvetítő felelős a kommunikációért, így az objektumok nem kommunikálnak közvetlenül egymással, hanem csak a mediátoron keresztül.

*Példa: Egy chat-szobában a felhasználók nem közvetlenül egymással kommunikálnak, hanem a chat-kiszolgálón keresztül.*

* Memento (Emlékeztető):

A Memento minta lehetővé teszi, hogy egy objektum állapotát elmentsük és később visszaállítsuk. Ez hasznos, ha például egy alkalmazás visszavonási funkciót kínál, mivel el lehet menteni az állapotokat különböző időpontokban.

*Példa: Egy grafikai programban az "undo" funkció használata, ahol az objektum korábbi állapotait mentjük és szükség esetén visszaállítjuk.*

* Observer (Megfigyelő):

Az Observer minta egy „közzétevő-előfizető” modellt használ. Egy objektum (az alany) értesítést küld a hozzá kapcsolódó megfigyelőknek, amikor annak állapota megváltozik. Ez lehetővé teszi, hogy a megfigyelők automatikusan frissüljenek az alany állapotváltozásakor.

*Példa: Egy felhasználói felület, ahol a gomb megnyomásakor több más elem is frissül.*

* State (Állapot):

A State minta lehetővé teszi, hogy egy objektum a belső állapota alapján változtassa a viselkedését, mintha az objektum osztálya dinamikusan változna. Ez segít az állapotgépek implementálásában.

*Példa: Egy videólejátszó, amely a lejátszás, megállítás vagy szünetelés állapota alapján más-más viselkedést mutat.*

* Strategy (Stratégia):

A Strategy minta lehetővé teszi, hogy egy algoritmust futásidőben válasszunk ki azáltal, hogy egy interfészen keresztül különböző algoritmusokat definiálunk. A minta célja, hogy az algoritmusokat egymástól függetlenül cserélgethessük anélkül, hogy az algoritmust használó kódot módosítanánk.

*Példa: Egy fizetési rendszer, amely különböző fizetési stratégiákat (hitelkártya, PayPal, kriptovaluta) támogat.*

* Template Method (Sablon metódus):

A Template Method minta lehetővé teszi, hogy egy művelet keretrendszerét definiáljuk egy absztrakt osztályban, majd a konkrét lépéseket a leszármazott osztályokban implementáljuk. Ez a minta elősegíti a kód újrahasználhatóságát, mert az általános logika egy központi helyen van definiálva.

*Példa: Egy adatfeldolgozó rendszer, ahol a feldolgozási lépések általánosak (pl. betöltés, feldolgozás, mentés), de a konkrét részletek (pl. adatok formátuma) minden alosztályban eltérőek lehetnek.*

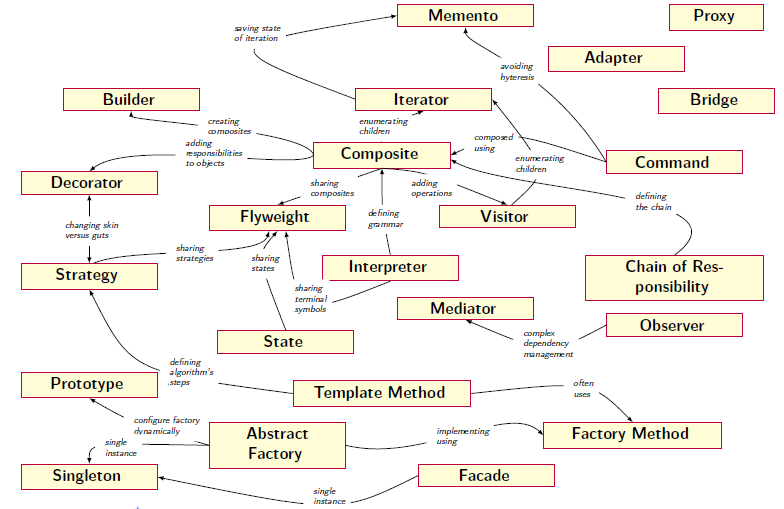
* Visitor (Látogató):

A Visitor minta lehetővé teszi, hogy egy műveletet különböző típusú objektumokra alkalmazzunk anélkül, hogy módosítanánk az osztályaikat. Ez megkönnyíti új műveletek hozzáadását az objektumokhoz anélkül, hogy módosítanánk magukat az objektumokat.

*Példa: Egy fájlrendszer elemeinek (fájlok és mappák) különböző műveletei, mint például méret kiszámítása vagy fájl formátumának átalakítása, külön kezelhetők egy Visitor implementációval.*

Minták közötti kapcsolatok:

Sokszor a minták nem léteznek önmagukban, használatuk egyéb minták használatát igényelheti. Ezeket a kapcsolatokat szemlélteti az alábbi ábra:



* **Miért hasznosak a tervezési minták?**
* Újrahasználhatóság: A tervezési minták elősegítik az újrahasználható kódrészletek létrehozását, ami időt és erőforrásokat takarít meg a fejlesztés során.
* Rugalmasság: A tervezési minták segítenek abban, hogy a kód könnyebben karbantartható és módosítható legyen anélkül, hogy a teljes rendszer újraírására lenne szükség.
* Olvashatóság: A minták használata növeli a kód olvashatóságát, mivel egy-egy jól ismert megoldás alkalmazása könnyebben érthető más fejlesztők számára is.
* Skálázhatóság: A megfelelő minták használatával a programok könnyedén bővíthetők új funkciókkal vagy viselkedésekkel.
* **Példák Java-ban:**

Java nyelven számos tervezési mintát megvalósíthatunk.

Például:

* Singleton minta: Egy osztály csak egy példányt hoz létre, például egy adatbázis kapcsolat kezelésére.
* Factory Method: A GUI elemek létrehozása különböző platformokon.
* Observer minta: Értesítési rendszerek, ahol egy objektum változásait figyelik a megfigyelők, például grafikus felületeken.

1. **MVC mint modell nézet vezérlő minta és néhány másik tervezési minta:**

**MVC (Model-View-Controller) minta:**

Az MVC minta három fő komponensre osztja az alkalmazást: Model, View, és Controller.

Ez az elkülönítés segít abban, hogy a különböző részek függetlenül fejleszthetők legyenek, és megkönnyíti a kód karbantartását és tesztelhetőségét.

* Model:

A modell felelős az adatok kezeléséért, tárolásáért és üzleti logikájáért. Ez a komponens az alkalmazás állapotát reprezentálja, és frissíti a nézetet, amikor az adatok változnak.

*Példa: Egy e-kereskedelmi alkalmazásban a termékek adatainak tárolásáért és a készletkezelésért a Model komponens felelős.*

* View:

A nézet az alkalmazás felhasználói felületét biztosítja, és megjeleníti az adatokat a felhasználó számára. A View a Model adataira reagál, de nem tartalmaz logikát vagy üzleti szabályokat.

*Példa: Az e-kereskedelmi alkalmazásban a View megjeleníti a terméklistát, a felhasználó által végzett keresés eredményeit, és a termékek részleteit.*

* Controller:

A vezérlő kezeli a felhasználói beviteleket, frissíti a modellt és meghatározza, hogy melyik nézet jelenjen meg. A Controller összekapcsolja a nézetet és a modellt.

*Példa: Amikor a felhasználó megnyom egy gombot, a Controller eldönti, hogy a Model melyik adatait kell frissíteni, és hogy melyik View-t kell megjeleníteni.*

Az MVC minta alkalmazása különösen hasznos webalkalmazásokban, ahol a felhasználói interakciók gyorsan és könnyen módosíthatók a logikai réteg vagy a megjelenítés megváltoztatása nélkül.

**További tervezési minták:**

* Observer minta:

Az Observer minta egy "közzétevő-előfizető" modellre épül, ahol az egyik objektum (közzétevő vagy alany) értesítést küld az őt megfigyelő (előfizető) objektumoknak, amikor valamilyen változás történik. Ez a minta biztosítja, hogy a különböző objektumok értesüljenek a változásokról anélkül, hogy azok szorosan kapcsolódnának egymáshoz.

*Példa: Egy hírportál rendszere, ahol a felhasználók különböző kategóriákra (pl. sport, politika) iratkozhatnak fel, és amikor egy új hír érkezik, a megfigyelők (feliratkozók) értesítést kapnak.*

* Singleton minta:

A Singleton minta célja annak biztosítása, hogy egy osztálynak csak egyetlen példánya legyen az alkalmazásban, és egy globális hozzáférési pontot biztosít ehhez a példányhoz. Ez különösen hasznos olyan helyzetekben, amikor egy erőforrás-igényes objektumot csak egyszer szeretnénk létrehozni, például adatbázis kapcsolat vagy naplózási rendszerek esetén.

*Példa: Egy adatbázis kapcsolat kezelése egy webalkalmazásban. Csak egy adatbázis-kapcsolat példány jön létre, és minden felhasználó ugyanazt a kapcsolatot használja.*

* Factory Method minta:

A Factory Method minta egy olyan tervezési minta, amely egy interfészt biztosít objektumok létrehozásához, de a konkrét példányosítást a leszármazott osztályokra bízza. Ez a minta lehetővé teszi, hogy az osztályok ne legyenek szorosan kötve a létrehozott objektumaikhoz, és így a rendszer rugalmasabb és könnyebben bővíthető legyen.

*Példa: Egy játékban különböző szörnyeket szeretnénk létrehozni, de a szörnyek típusai csak futásidőben derülnek ki. A Factory Method minta lehetővé teszi a szörnyek dinamikus létrehozását a megfelelő osztály segítségével.*

* Strategy minta:

A Strategy minta célja, hogy egy algoritmust vagy műveletet futásidőben válasszunk ki, különválasztva az algoritmus interfészét és annak konkrét implementációit. Ez lehetővé teszi az algoritmusok közötti egyszerű váltást, anélkül, hogy az osztályok logikáját módosítani kellene.

*Példa: Egy online fizetési rendszer, amely különböző fizetési stratégiákat (pl. PayPal, hitelkártya, kriptovaluta) támogat. A fizetési stratégia dinamikusan választható ki a felhasználó választása alapján.*

Források:

<https://okt.inf.szte.hu/prog1/gyakorlat/eloadas/TervezesiMintak/mintakAttekintese/>

https://mernokinformatikus.hu/tervezesi-mintak-a-gyakorlatban/

<https://www.geeksforgeeks.org/complete-guide-to-design-patterns-in-programming/>

<https://www.tutorialspoint.com/design_pattern/index.htm>

<https://refactoring.guru/design-patterns>