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

### Bevezetés

A tervezési minták olyan útmutatók és újrafelhasználható megoldások, amelyek segítik az objektumorientált (OO) programok tervezését és strukturálását. Az OO programozás alapelve az, hogy az alkalmazást olyan objektumokból építjük fel, amelyek egymással kommunikálnak. A tervezési minták ezt a kommunikációt, az osztályok közötti kapcsolatokat és a kódstruktúrát segítik optimalizálni.

### 1. Az Objektumorientált Programozás Alapelvei

Az OO programozás alapelvei:

#### 1.1 Egységbezárás elve (Single Responsibility Principle - SRP)

Az SRP szerint egy osztálynak csak egy oka legyen a változásra. Egy osztálynak egyetlen felelőssége kell, hogy legyen.

#### 1.2 Nyitott/Zárt elve (Open/Closed Principle - OCP)

Az OCP elve szerint egy osztály zárt a módosításokra, de nyitott az kiterjesztésekre. Új funkciókat lehet hozzáadni, anélkül hogy a meglévő kódot módosítanánk.

#### 1.3 Liskov helyettesítési elve (Liskov Substitution Principle - LSP)

Az LSP elve szerint az alkalmazásban bármelyik leszármazott osztály helyettesíthető az ősosztállyal anélkül, hogy megváltoztatná az alkalmazás helyes működését.

#### 1.4 Interfész szegregáció elve (Interface Segregation Principle - ISP)

Az ISP elve szerint egy osztálynak nem kell olyan interfészt implementálnia, amelynek funkciói nem relevánsak az osztály számára.

#### 1.5 Függőségi inverzió elve (Dependency Inversion Principle - DIP)

Az DIP elve szerint az absztrakcióknak nem kell függeniük a részletektől, hanem fordítva. A magas szintű moduloknak nem kell függniük a részletektől, hanem azoknak a magas szintű moduloknak.

### 2. Tervezési Minták általánosságban

A tervezési minták széles körben alkalmazhatóak az OO programozási nyelvekben. Ezek segítenek a kód újrafelhasználhatóságának növelésében, a rugalmasság megteremtésében és a karbantarthatóság javításában.

#### 2.1 Alapelvek és Tervezési Minták Kapcsolata

A SOLID alapelvek és a tervezési minták szorosan kapcsolódnak egymáshoz. A tervezési minták segítenek megvalósítani az OO alapelveket a gyakorlatban.

#### 2.2 Néhány Fontos Tervezési Minta

##### 2.2.1 Osztályszintű minták

* **Egyszerű Gyár (Simple Factory):** Segíti az objektumok létrehozását egy közös interfész révén.
* **Absztrakt Gyár (Abstract Factory):** Biztosít egy interfészt az objektumok családjának létrehozásához anélkül, hogy meg kellene változtatni az alkalmazást.

##### 2.2.2 Objektumszintű minták

* **Stratégia Minta (Strategy):** Lehetővé teszi egy algoritmus cseréjét úgy, hogy az nem befolyásolja a kliens osztályt.
* **Létrehozó Minta (Builder):** Segít egy összetett objektum létrehozásában, lépésről lépésre.

##### 2.2.3 Struktúrális Minták

* **Dekorátor Minta (Decorator):** Lehetővé teszi egy objektum funkcionalitásának bővítését, dinamikusan.
* **Fasad Minta (Facade):** Egy egyszerű interfészt biztosít egy összetett rendszerhez.

#### 2.3 Tapasztalatok és Javaslatok

A tervezési minták használatakor fontos figyelembe venni a konkrét alkalmazás igényeit. A túlzott tervezési minta használat felesleges bonyolultságot és rugalmatlanságot eredményezhet.

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

A tervezési minták olyan útmutatók, amelyek segítik az objektumorientált (OO) programok kialakítását. Az OO nyelvek nagyrészt a modell-nézet-vezérlő (MVC) mintát használják, de számos más tervezési minta is létezik. Ez a tétel részletesen foglalkozik az MVC mintával és bemutat néhány másik jelentős tervezési mintát egy OO programozási nyelvben.

### 1. Modell-Nézet-Vezérlő (MVC) Minta

#### 1.1 Modell

A Modell az MVC minta szívében található komponens, amely az alkalmazás belső reprezentációjáért felel. Itt találhatók azok az osztályok és adatstruktúrák, amelyek az alkalmazás állapotát tárolják és manipulálják. A Modell tartalmazza a vállalkozás logikáját, üzleti szabályokat és adatelérési réteget. Például, egy webalkalmazásban a Modell lehet a felhasználók, termékek és tranzakciók kezelését végző osztályok összessége.

A Modell az OO tervezési elvekkel összhangban megvalósítja az Egységbezárás elvét (SRP), mivel csak egy felelősséggel bír: az alkalmazás állapotának kezelésével.

#### 1.2 Nézet

A Nézet a MVC minta második pillére, amely az adatok megjelenítéséért és a felhasználói felületért felel. Ez az interfész azonosítja azokat a komponenseket, amelyekkel a felhasználó közvetlenül interakcióba lép. A Nézetek kizárólag a felhasználói felülettel foglalkoznak, nem tartalmazzák az üzleti logikát vagy az adatok manipulációját.

A Nézetek a gyakorlatban lehetnek például weboldalak, grafikus felhasználói felületek vagy mobilalkalmazások. Ezeknek a komponenseknek a létrehozása, frissítése és eltávolítása lehetőséget ad az alkalmazás dinamizmusára és könnyű alkalmazkodására a változó felhasználói igényekhez.

#### 1.3 Vezérlő

A Vezérlő az a komponens, amely a felhasználói interakciókat kezeli és irányítja. Ez az elem kapja meg a felhasználói inputot a Nézetekből, majd továbbítja azt a Modellnek a szükséges műveletek elvégzése érdekében. A Vezérlő felelős az alkalmazás működésének szabályozásáért, és biztosítja, hogy a Nézet és a Modell közötti kommunikáció szigorúan elkülönüljön.

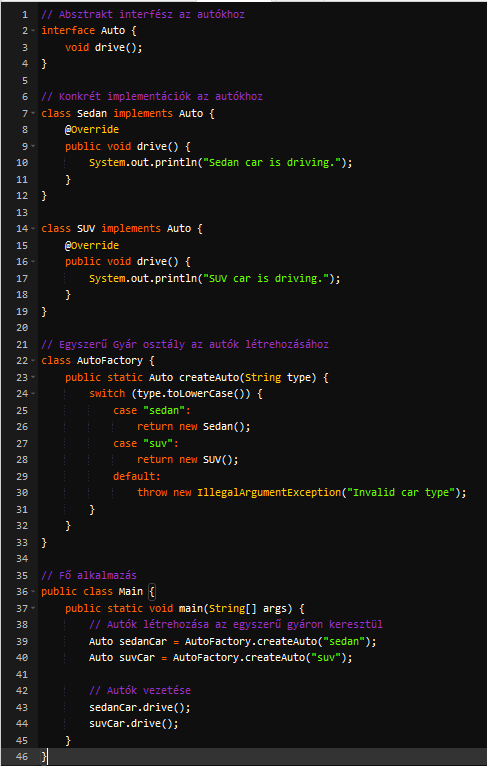
A Vezérlő az alkalmazásban megtestesíti az Interfész Szegregáció Elvét (ISP), mivel csak azokat a funkciókat tartalmazza, amelyek a felhasználói interakciókat kezelik, nem terhelve le az alkalmazást olyan felesleges funkciókkal, amelyek nélkülözhetők lehetnek.

### 2. Példák a Gyakorlatban

### Példa: Egyszerű Gyár (Simple Factory)

Az Egyszerű Gyár minta egy olyan tervezési minta, amely segít az objektumok létrehozásában egy közös interfész révén. Vegyük például egy autógyártási alkalmazást, ahol különböző típusú autókat szeretnénk létrehozni.

Ebben a példában az Auto interfész definiálja az autók közös viselkedését, míg a Sedan és SUV osztályok megvalósítják ezt az interfészt. Az AutoFactory osztály az autók létrehozásáért felel, és az alkalmazásban csak ezt a gyárat kell használnunk az autók létrehozásához.



### Példa: Stratégia (Strategy)

A Stratégia minta lehetővé teszi, hogy egy algoritmus cserélhető legyen a kliens osztály számára anélkül, hogy az kliens osztály kódját módosítanánk. Vegyük például egy fizetési rendszer alkalmazását, ahol különböző fizetési stratégiákat szeretnénk támogatni.

Ebben a példában a PaymentStrategy interfész definiálja a fizetési stratégiák közös viselkedését, míg a CreditCardPayment és PayPalPayment osztályok megvalósítják ezt az interfészt. A ShoppingCart osztály használja a kiválasztott fizetési stratégiát a fizetési műveletek során.

