

Nyíregyházi Egyetem

PTI szak

**Tervezési minták egy objektumorientált
programozási nyelvben**

Tantárgy kódja: BPI1116L

Tanév: 2025/26 félév: 1

Készített:

Név: Bálint Botond

Neptun kód: FLOCRM

Szak: Programtervező informatikus

Tartalom

1. Bevezetés:	3
2. A tervezési minták szerepe és előnyei.....	3
3. Az MVC – Model–View–Controller minta	4
3.1 Model (Modell)	4
3.2 View (Nézet)	5
3.3 Controller (Vezérlő).....	5
4. A Singleton létrehozási minta	6
4.1 A Singleton célja:.....	6
4.2 Működési elve és a minta lényege:	6
5. További minták rövid áttekintése:.....	7
5.1 Factory Method (létrehozási minta):.....	7
5.2 Strategy (viselkedési minta):	7
5.3 Adapter (szerkezeti minta):	7
6. Összegzés:.....	8
7. Felhasznált irodalom:	9

1. Bevezetés:

A szoftverfejlesztés története során a programozók újra és újra szembesültek ugyanazokkal a problémákkal: hogyan lehet a kódot úgy felépíteni, hogy az egyszerre legyen átlátható, módosítható, bővíthető és újrahasznosítható. Ezekre a kihívásokra adnak választ a **tervezési minták**, amelyek általános, jól bevált megoldásokat kínálnak gyakran előforduló programozási problémákra. A minták nem konkrét kódrészletek, hanem olyan szoftver architekturális irányelvek és szerkezeti sémák, amelyeket különböző objektumorientált nyelvekben egyaránt alkalmazni lehet.

A tervezési minták azért fontosak, mert segítenek abban, hogy a program jól fel legyen építve: a kód kisebb, könnyen kezelhető részekre bontható, minden résznek világos feladata van, a módosítások könnyebben elvégezhetők, és így kevesebb hiba keletkezik.

A fejlesztők a minták használatával elkerülhetik a gyakori tervezési hibákat és olyan megoldásokhoz juthatnak, amelyek széles körben elfogadottak.

Kiváló összefoglalót nyújt több minta működéséről a *refactoring.guru* oldal. A minták áttekintését és a programozói „best practice”-ek bemutatását az alábbiakban is többek között innen feldolgozott gondolatok is kiegészítik, különösen a Singleton minta esetében (Forrás: refactoring.guru/design-patterns/singleton).

2. A tervezési minták szerepe és előnyei

Az objektumorientált fejlesztés egyik kulcsa az, hogy az alkalmazást jól elkülönülő részekre lehet bontani. A tervezési minták ebben segítenek: szabályokat adnak arra, hogyan kommunikáljanak egymással az objektumok, hogyan jöjjenek létre példányok, és hogyan legyenek a felelőségek ésszerűen szétosztva.

A tervezési minták használatának néhány előnye:

- **Jobb karbantarthatóság:** a kód könnyebben átlátható és módosítható.
- **Újrafelhasználhatóság:** ugyanaz a szerkezeti megoldás több projektben is alkalmazható.
- **Egységes megoldások:** a szakmában elfogadott sablonokról minden fejlesztő tudja, mit jelentenek.
- **Csökkentett hibalehetőség:** a minta logikája előre lefektetett, így kisebb az esély a szerkezeti hibákra.

- **Laza csatolás:** az objektumok kevésbé függenek egymástól, ami rugalmasságot eredményez.

A hazai egyetemi jegyzetek például az ELTE és az SZTE tervezési mintákat bemutató anyagai szintén hangsúlyozzák, hogy a minták egyik fő célja a komplexitás kezelése és a program jól definiált szerkezeti felépítése.

3. Az MVC – Model–View–Controller minta

Az egyik legismertebb és leggyakrabban használt architektúrális minta az **MVC** (Model-View-Controller). Ezt a mintát eredetileg asztali alkalmazások grafikus felületeinek kezelésére dolgozták ki, de mára rendkívül elterjedt a webfejlesztésben, mobilalkalmazásokban és egyéb interaktív rendszerekben is.

Az MVC célja, hogy **szétválassza a program felelősségi köreit három jól elkülönülő részre:** az adatok kezelésére, a megjelenítésre és a felhasználói interakciók kezelésére.

A három komponens szerepe:

3.1 Model (Modell)

A modell felelős az adatokért és az üzleti logikáért. Ez a program „lelke”: itt találhatóak az osztályok, amelyek kezelik az adatbázis-műveleteket, a számításokat, az állapotváltozásokat. A modell csak az adatokkal foglalkozik nem tud róla, hogy a felhasználó mit lát a képernyőn. Ez a felelősségszétválasztás alapvető jó gyakorlata. A megjelenítés és az üzleti logika nem keveredik össze.

3.2 View (Nézet)

A nézet feladata kizárólag az adatok **megjelenítése**. A nézet a modell által szolgáltatott adatokból felépíti a felhasználói felületet.

Webes környezetben ez lehet egy HTML oldal sablonnal, asztali alkalmazásban pedig egy grafikus panel.

Fontos, hogy a nézet nem tartalmaz üzleti logikát. Ez megakadályozza, hogy a kód később kezelhetetlenné váljon.

3.3 Controller (Vezérlő)

A vezérlő közvetíti a felhasználó és az alkalmazás között.

Feladata:

- fogadja a felhasználói eseményeket (kattintás, űrlapbeküldés, stb.),
- utasítja a modellt, hogy frissítse az adatokat,
- majd értesíti a nézetet, hogy frissítse a felületet.

A vezérlő tehát a rendszer „irányítója”, amely futásidőben összehangolja a modell és a nézet működését.

Az MVC előnyei

Az MVC, akárcsak más minták, jól illeszkedik a moduláris tervezésbe:

- **Átláthatóbb kód:** minden rétegnek világos szerepe van.
- **Könnyebb fejlesztés:** akár több fejlesztő tud párhuzamosan dolgozni (egyik a nézeten, másik a modellen, stb.).
- **Rugalmasság:** a megjelenítés változtatható anélkül, hogy a logikához hozzá kellene nyúlni.
- **Tesztelhetőség:** a modell külön is tesztelhető, mivel nem függ a felülettől.

Ezek a szempontok gyakran szerepelnek magyar egyetemi programozási jegyzetekben is mint ajánlott gyakorlatok.

4. A Singleton létrehozási minta

A tervezési minták egyik legismertebb csoportja a **létrehozási minták (creational patterns)**, amelyek arra adnak megoldást, hogyan és milyen módon jöjjenek létre objektumok egy alkalmazásban.

A legismertebb közülük a **Singleton**. A definíciót és a működési elvet kiválóan mutatja be a refactoring.guru oldalon található leírás.

4.1 A Singleton célja:

A Singleton olyan osztályt ír le, amelyből **csak egyetlen példány** létezhet, és ezt globálisan elérhetővé teszi.

Ez többek között akkor hasznos, ha:

- adatbázis-kapcsolatot kezelünk,
- központi naplózási rendszert működtetünk,
- konfigurációs beállításokat szeretnénk egységesen kezelni.

4.2 Működési elve és a minta lényege:

1. Az osztály konstruktora **privát**, így kívülről nem példányosítható.
2. Van egy **statikus metódus**, például `getInstance()`.
3. Ez a metódus ellenőrzi, hogy létezik-e már példány:
 - ha nem, akkor létrehozza,
 - ha igen, akkor visszaadja a már létezőt.

Ez megoldás egyszerre lehet előny és veszélyforrás is: könnyen vezethet túlzottan globális függésekhez, ezért óvatosan kell használni. A minta akkor jó választás, ha valóban csak egyetlen központi erőforrást kell kezelni.

5. További minták rövid áttekintése:

A tervezési minták három fő csoportba sorolhatók:

- **Létrehozási minták** (pl. Singleton, Factory Method, Builder)
- **Szerkezeti minták** (pl. Adapter, Facade)
- **Viselkedési minták** (pl. Strategy, Observer)

5.1 Factory Method (létrehozási minta):

A Factory Method célja, hogy a példányosítást egy külön metódusba szervezze ki. Így a konkrét objektumtípus létrehozása rugalmasan változtatható anélkül, hogy a kód többi részéhez hozzá kellene nyúlni.

5.2 Strategy (viselkedési minta):

A Strategy akkor használatos, ha egy algoritmust többféle módon lehet megvalósítani. A minta lehetővé teszi, hogy ezeket az algoritmusokat egymással felcserélhető, elkülönített osztályokban tároljuk, így a futásidőben is cserélhetők.

5.3 Adapter (szerkezeti minta):

Feladata, hogy két olyan osztályt kapcsoljon össze, amelyek eredeti formájukban nem kompatibilisek egymással. Olyan „átalakítót” biztosít, amely összehangolja az eltérő felületeket.

Ezek a minták segítenek a kód bővíthetőségében és újrafelhasználhatóságában, és gyakran említve vannak a szakirodalomban, mint ajánlott szerkezeti megoldások.

6. Összegzés:

A tervezési minták alkalmazása számottevően javítja a szoftverek minőségét, karbantarthatóságát és élettartamát. Bár maguk a minták nem helyettesítik a jó tervezést, mégis olyan irányelveket adnak, amelyek a fejlesztési folyamatban támpontot nyújtanak a strukturált és átgondolt megoldásokhoz.

Az MVC különösen jól mutatja, milyen előnyökkel jár a felelősségek világos szétválasztása: külön réteg kezeli az adatokat, külön a megjelenítést, és külön a felhasználói interakciók logikáját. Ez a minta a grafikus felületek és webes rendszerek egyik legelterjedtebb architektúrája.

A Singleton és más tervezési minták pedig olyan alapvető szoftverépítési problémákra adnak megoldást, mint az objektumok létrehozása, a laza csatolás vagy a cserélhető algoritmusok kezelése.

A tervezési minták tehát nem dogmák, hanem eszközök: akkor működnek jól, ha megfelelő helyen és megfelelő céllal alkalmazzuk őket. Az objektumorientált fejlesztésben azonban ma már szinte nélkülözhetetlenek.

7. Felhasznált irodalom:

- **refactoring.guru – „Singleton” minta leírása és működése**
<https://refactoring.guru/design-patterns/singleton>
- **refactoring.guru – Design Patterns katalógus, mintaáttekintések**
<https://refactoring.guru/design-patterns>
- **ELTE informatika jegyzet – Tervezési minták áttekintése**
(people.inf.elte.hu - tervezési minták oktatási anyaga)
- **SZTE programozási minták összefoglalója**
(okt.inf.szte.hu - Tervezési minták előadásanyag)