

Objektum-orientált tervezési minták

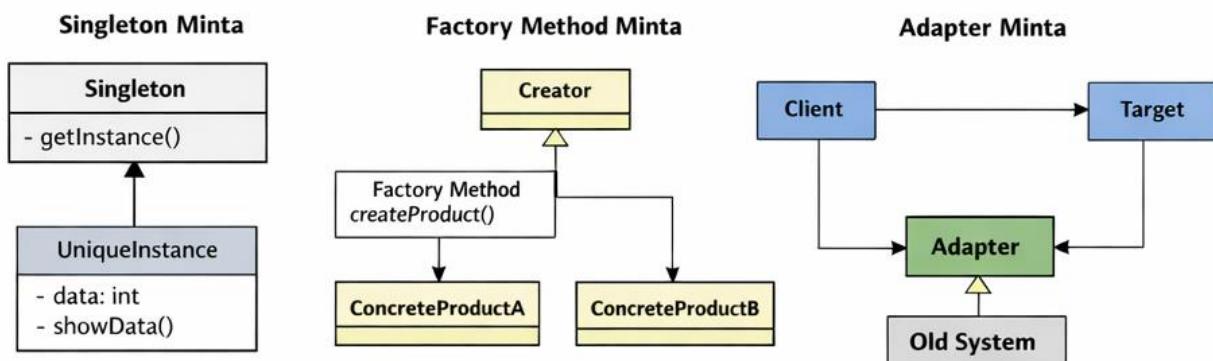
1. Bevezetés

Az objektum-orientált tervezési minták (Design Patterns) bevált megoldások a szoftvertervezési problémákra. Céljuk:

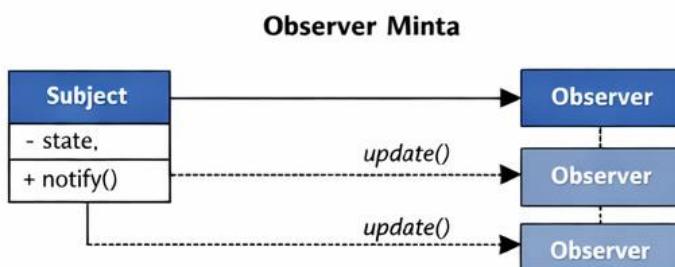
- Növelik a kód újrafelhasználhatóságát.
- Könnyítik a kommunikációt a fejlesztések között.
- Javitják a szoftver karbantarthatóságát.

2. A tervezési minták csoportosítása

Tipus	Leírás	Példa
Kreációs minták	Példányositás megoldásai	<i>Singleton, Factory Method</i>
Szerkezeti minták	Objektumok struktúrájának kezelése	<i>Adapter, Composite</i>
Viselkedési minták	Kommunikáció és viselkedés	<i>Observer, Strategy</i>



3. Viselkedési Minta Példa



4. Összegzés

A tervezési minták segítik a kód újrahasznosítását, karbantarthatóságát és a fejlesztési folyamatot.

Az objektum-orientált programozás (OOP) célja, hogy a szoftverfejlesztést modulárisabbá, könnyebben karbantarthatóvá és újrahasznosíthatóvá tegye. Azonban a komplex rendszerek fejlesztése során gyakran előfordulnak ismétlődő tervezési problémák. Az **objektum-orientált tervezési minták** (Design Patterns) olyan bevált megoldások, amelyek ezekre a problémákra nyújtanak rendszerezett és kipróbált módszereket.

A tervezési minták célja, hogy:

- Növeljék a kód újrafelhasználhatóságát.
- Könnyítsék a kommunikációt a fejlesztők között (egy közös "nyelvet" biztosítanak).
- Csökkentsék a hibák előfordulását és javítsák a szoftver karbantarthatóságát.

A legismertebb forrás a "**Gang of Four**" (**GoF**) **könyv**, amely 23 alapvető tervezési mintát különböztet meg.

A tervezési minták csoportosítása

A GoF tervezési minták három fő kategóriába sorolhatók:

1. Kreációs minták (Creational Patterns)

Ezek a minták a **példányosítás problémáira** adnak megoldást. Segítenek abban, hogy az objektumok létrehozása rugalmas, moduláris és független legyen a konkrét osztályuktól.

Fő kreációs minták:

- **Singleton:** Biztosítja, hogy egy osztálynak csak egy példánya legyen, és globális hozzáférést biztosít hozzá.

Példa: Konfigurációs beállítások kezelése.

A Singleton minta egy osztály egyetlen példányát biztosítja, amelyhez globális hozzáférés biztosítható. Gyakran használják olyan erőforrások kezelésére, amelyeknek centralizáltnak kell lenniük, például konfigurációs beállítások, naplózók vagy adatbázis-kapcsolatok.

1. Tulajdonságok:

1. Egyetlen példány létezik az osztályból.
2. Globális hozzáférés a példányhoz.
3. Kontrollált példányosítás (késleltetett inicializálás).
4. Többszálú alkalmazásoknál thread-safe implementáció szükséges.

- **Factory Method:** Egy interfészen keresztül hoz létre objektumokat anélkül, hogy a konkrét osztályt meg kellene határozni.

Példa: Különböző típusú dokumentumok létrehozása egy szerkesztőben.

- **Abstract Factory:** Kapcsolódó objektumok családjának létrehozását teszi lehetővé anélkül, hogy az implementációt ismerni kellene.

Példa: GUI komponensek különböző operációs rendszerekhez.

- **Builder:** Bonyolult objektumok lépésenkénti létrehozása.

Példa: PDF vagy HTML dokumentum generálása.

A Builder minta komplex objektumok lépésenkénti felépítésére szolgál, elválasztva az objektum konstrukcióját a reprezentációtól. Ez lehetővé teszi ugyanannak az építési folyamatnak különböző kimenetek létrehozását.

1. Tulajdonságok:

1. Elválasztja az objektum építését és reprezentációját.
2. Lépésenkénti építés, amely rugalmas és bővíthető.

3. Különböző reprezentációk létrehozására alkalmas.

- **Prototype:** Objektumok klónozása meglévő példányok alapján.
Példa: Játék karakterek vagy konfigurációs sablonok másolása.

2. Szerkezeti minták (Structural Patterns)

Ezek a minták objektumok és osztályok összekapcsolására, szerkezetük kezelésére adnak megoldást.

Fő szerkezeti minták:

- **Adapter:** Két inkompatibilis interfész összekapcsolása.
Példa: Régi API-k új rendszerekhez való illesztése.
- **Bridge:** Absztrakciók és implementációk szétválasztása, hogy külön-külön változtathatók legyenek.
Példa: Különböző rajzolási algoritmusok támogatása különböző eszközökön.
- **Composite:** Egy objektum és annak összetevői egységes módon kezelhetők.
Példa: Fájlstruktúra, ahol fájlok és mappák egyaránt kezelhetők.
- **Decorator:** Objektumok funkcionalitásának futásidőben történő bővítése.
Példa: Java I/O stream-ek kiegészítése.
- **Facade:** Egyszerűsített interfész biztosítása komplex alrendserekhez.
Példa: Egy összetett könyvtár egyszerű API-jának létrehozása.
- **Flyweight:** Nagy számú objektum memóriatakarékos kezelése.
Példa: Szövegszerkesztők karakterek objektumainak újrafelhasználása.
- **Proxy:** Egy objektum helyettesítése egy másik objektummal, amely kontrollálja az elérést.
Példa: Virtuális proxy képek betöltésére.

3. Viselkedési minták (Behavioral Patterns)

Ezek a minták az objektumok közötti kommunikációt és viselkedést segítik szabályozni.

Fő viselkedési minták:

- **Observer:** Egy objektum változásait több másik objektum követheti.
Példa: GUI eseménykezelés, híroldalak értesítése.
Az Observer minta lehetővé teszi, hogy egy objektum (Subject) értesítse a hozzá kapcsolódó objektumokat (Observers) a változásokról anélkül, hogy az értesített objektumok szoros kapcsolatban lennének a Subjecttel.
 1. Tulajdonságok:
 1. Laza kapcsolódás a Subject és Observer-ek között.
 2. Automatikus értesítés a változásokról.
 3. Könnyen bővíthető új Observer-ek hozzáadásával.
 2. Stratégiai minták:
 - **Strategy:** Különböző algoritmusok cserélhetők futásidőben.
Példa: Különböző fizetési módok egy webshopban.
 - **Command:** Műveletek objektumokként kezelése, visszavonás támogatásával.
Példa: Szövegszerkesztő visszavonás/funkciók.
 - **Chain of Responsibility:** Kérés feldolgozása láncolt objektumokon keresztül.
Példa: Hibakezelés több szinten.
 - **Mediator:** Objektumok közötti kommunikáció központi kezelése.
Példa: Chat szerver üzenetkezelése.

- **State:** Objektum viselkedése változik belső állapota szerint.
Példa: Játék karakter különböző állapotai (alvás, futás, támadás).
- **Template Method:** Algoritmus vázának meghatározása, bizonyos lépések implementációját a leszármazottakra bízza.
Példa: Adatfeldolgozás különböző adatforrásokból.
- **Iterator:** Objektumok sorozatának bejárása anélkül, hogy belső reprezentációjukat ismerni kellene.
Példa: Lista, tömb vagy gyűjtemény bejárása.
- **Visitor:** Új műveletek hozzáadása objektumstruktúrákhoz anélkül, hogy módosítani kellene az osztályokat.
Példa: Elemzés és riport készítés különböző objektumtípusokhoz.

Példa a gyakorlatban

Egy webáruházban például gyakran használhatjuk:

- **Singleton:** Adatbázis kapcsolat kezelése.
- **Factory Method:** Kosárba kerülő termékek példányosítása.
- **Observer:** Vásárlók értesítése, ha egy termék újra elérhető.
- **Strategy:** Fizetési módok kiválasztása (bankkártya, PayPal, utánvét).

Ez jól mutatja, hogy a tervezési minták **egymással kombinálhatók**, és segítik a kód olvashatóságát, bővíthetőségét.

Összegzés

Az objektum-orientált tervezési minták alapvető eszközök a szoftverfejlesztésben.

Előnyeik:

- Megkönnyítik a komplex rendszerek fejlesztését.
- Növelik a kód újrahasznosíthatóságát és karbantarthatóságát.
- Csökkentik a hibák előfordulását.

A tervezési minták ismerete és helyes alkalmazása a professzionális programozás egyik alappillére, amely hozzájárul a stabil és rugalmas szoftverek létrehozásához.