Tervezési minták egy OO programozási nyelvben. MVC, mint modell-nézet-vezérlő minta és néhány másik tervezési minta

A tervezési minták a szoftvertervezésben gyakran előforduló problémákra nyújtanak megoldásokat. Olyanok, mint az előre elkészített tervrajzok, amelyeket testre szabhatunk a kódban visszatérő tervezési problémák megoldása érdekében. A minta nem egy konkrét kódrészlet, hanem egy általános koncepció egy adott probléma megoldására.

A mintákat gyakran összekeverik az algoritmusokkal, mivel mindkét fogalom néhány ismert probléma tipikus megoldásait írja le. Míg egy algoritmus mindig egy műveletsort határoz meg, amellyel elérhet valamilyen célt, a minta a megoldás magasabb szintű leírása. Két különböző programra alkalmazott ugyanazon minta kódja eltérő lehet.

Legtöbb tervezési minta nagyon formálisan van leírva, hogy könnyen reprodukálhatóak legyenek.

Egy tervezési minta leírása a következőket tartalmazza:

- A minta szándéka röviden leírja a problémát és a megoldását is. (**Intent**)
- A motiváció jobban leírja a problémát és a megoldásokat, melyeket az adott minta lehetővé tesz. (**Motivation**)
- Az osztályok struktúrája megmutatja a minta egyes részeit és hogy ezek hogyan kapcsolódnak egymáshoz. (**Structure**)
- Kód példa egy ismertebb programozási nyelven könnyebbé teszi a minta mögöttes tartalmának megértését. (**Code example**)

A tervezési mintáknak három fő típusa létezik:

1. Létrehozási (**Creational**)
2. Strukturális (**Structural**)
3. Viselkedési (**Behavioral**)

Létrehozási minták:

A létrehozási tervezési minták különböző objektumlétrehozási mechanizmusokat valósítanak meg, melyek növelik a rugalmasságot és az újra felhasználhatóságot.

Az alábbi létrehozási tervezési minták léteznek:

- **Singleton:** Lehetővé teszi, hogy egy osztálynak csak egy példánya legyen, miközben globális hozzáférési pontot biztosít ehhez a példányhoz.
- **Builder:** Lehetővé teszi egy objektum különböző típusainak és reprezentációinak előállítását ugyanazzal a konstruktorral.
- **Abstract Factory:** Lehetővé teszi a kapcsolódó objektumok családjainak létrehozását azok konkrét osztályainak megadása nélkül.

- **Factory Method:** Interfészt biztosít az objektumok létrehozásához egy szuperosztályban, de lehetővé teszi az alosztályok számára, hogy módosítsák a létrehozandó objektumok típusát.
- **Prototype:** Lehetővé teszi a meglévő objektumok másolását anélkül, hogy a kódot az osztályoktól függővé tenné.

Strukturális minták:

A strukturális tervezési minták megmagyarázzák, hogyan lehet objektumokat és osztályokat nagyobb struktúrákká összeállítani, miközben ezek a struktúrák rugalmasok és hatékonyak maradnak.

Az alábbi strukturális tervezési minták léteznek:

- **Adapter:** Lehetővé teszi az inkompatibilis interfészekkel rendelkező objektumok együttműködését.
- **Bridge:** Lehetővé teszi egy nagy osztály vagy szorosan kapcsolódó osztályok halmazának felosztását két különálló hierarchiára – absztrakcióra és megvalósításra –, amelyek egymástól függetlenül fejleszthetők.
- **Composite:** Lehetővé teszi, hogy az objektumokat fastruktúrákba állítsa össze, majd úgy dolgozzon ezekkel a struktúrákkal, mintha egyedi objektumok lennének.
- **Decorator:** Lehetővé teszi új viselkedések hozzárendelését az objektumokhoz azáltal, hogy ezeket az objektumokat a viselkedést tartalmazó speciális burkoló objektumokba helyezi.
- **Facade:** Egyszerűsített interfészt biztosít egy könyvtárhoz, keretrendszerhez vagy bármely más összetett osztálykészlethez.
- **Flyweight:** Lehetővé teszi, hogy több objektumot illesszen be a rendelkezésre álló RAM-mennyiségbe azáltal, hogy több objektum között megosztja a közös állapotrészeket, ahelyett, hogy az összes adatot megtartaná az egyes objektumokban.
- **Proxy:** Lehetővé teszi egy másik objektum számára helyettesítő vagy helyőrző biztosítását. A proxy vezérli az eredeti objektumhoz való hozzáférést, lehetővé téve, hogy végrehajtson valamit, mielőtt vagy miután a kérés eljut az eredeti objektumhoz.

Viselkedési minták:

A viselkedési tervezési minták az algoritmusokkal és az objektumok közötti felelősségek kiosztásával kapcsolatosak.

Az alábbi viselkedési tervezési minták léteznek:

- **Chain of Responsibility:** Lehetővé teszi, hogy kéréseket továbbítson a kezelők láncán. A kérés beérkezésekor minden kezelő eldönti, hogy feldolgozza-e a kérést, vagy továbbítja azt a lánc következő kezelőjének.

- **Command:** A kérést önálló objektummá alakítja, amely tartalmazza a kéréssel kapcsolatos összes információt. Ez az átalakítás lehetővé teszi a kérések metódusargumentumokként történő átadását, a kérés végrehajtásának késleltetését vagy sorba állítását, valamint a visszavonhatatlan műveletek támogatását.
- **Iterator:** Lehetővé teszi a gyűjtemény elemeinek bejárását anélkül, hogy felfedné a mögöttes reprezentációját (lista, verem, fa stb.).
- **Mediator:** Lehetővé teszi az objektumok közötti kaotikus függőségek csökkentését. A minta korlátozza az objektumok közötti közvetlen kommunikációt, és arra kényszeríti őket, hogy csak egy közvetítő objektumon keresztül működjenek együtt.
- **Memento:** Lehetővé teszi egy objektum korábbi állapotának mentését és visszaállítását anélkül, hogy felfedné a megvalósítás részleteit.
- **Observer:** Lehetővé teszi egy előfizetési mechanizmus meghatározását, amely több objektumot értesít minden olyan eseményről, amely az általuk megfigyelt objektummal történik.
- **State:** Lehetővé teszi az objektum viselkedésének megváltoztatását, ha belső állapota megváltozik. Úgy tűnik, mintha az objektum megváltoztatta volna az osztályát.
- **Strategy:** Lehetővé teszi, hogy meghatározzon egy algoritmuscsaládot, mindegyiket külön osztályba helyezze, és az objektumaikat felcserélhetővé tegye.
- **Template Method**: Meghatározza egy algoritmus vázát a szuperosztályban, de lehetővé teszi, hogy az alosztályok felülírják az algoritmus bizonyos lépéseit anélkül, hogy megváltoztatnák annak szerkezetét.
- **Visitor:** Lehetővé teszi az algoritmusok elkülönítését azoktól az objektumoktól, amelyeken működnek.