Tervezési minták egy OO programozási nyelvben

Készítette: Dandóczi Máté Attila
Neptun: AYIE0M

A tervezési minták olyan ismételhető, általános megoldási sémák, amelyek segítik a fejlesztőket a kód strukturálásában és szervezésében. Az objektumorientált (OO) programozásban ezek a minták különösen hatékonyak, mivel segítenek egy rugalmas, karbantartható és könnyen érthető kód kialakításában. Ebben a dokumentumban a Modell-Nézet-Vezérlő (MVC) minta mellett néhány másik tervezési mintát is megvizsgálunk.

Creational kreatív minták ötleteket adnak az objektum létrehozási mechanizmusokra, amelyekkel flexibilis és újrafelhasználható kódot kapunk.

Structural szerkezeti minták segítségével az osztályainkat rendszerezni tudjuk.

Behavioral viselkedésminták felosztják az objektumok közötti feladatokat az objektumok között.

Modell-Nézet-Vezérlő (MVC) Minta:

Az MVC minta egy strukturális tervezési elv, amely hatékonyan szétválasztja egy alkalmazás részeit a könnyebb karbantarthatóság és bővíthetőség érdekében.

Komponensek:

- Modell (Model): Az adatok és üzleti logika reprezentációja.
- Nézet (View): A felhasználói felület megjelenítéséért felelős rész.
- Vezérlő (Controller): Közvetíti a kapcsolatot a Modell és a Nézet között.

Működés:

1. A felhasználói interakció a Nézettel kezdődik.
2. A Nézet küldi az eseményeket a Vezérlőnek.
3. A Vezérlő feldolgozza az eseményeket és frissíti a Modellt vagy a Nézetet.
4. A Modell tárolja az adatokat, míg a Nézet megjeleníti azokat.

Előnyök:

- Független komponensek, könnyen fejleszthetők és tesztelhetők.
- Újrafelhasználhatóság: Modell vagy Nézet komponensek újrafelhasználhatók más alkalmazásokban.
- Könnyű karbantarthatóság: A kód módosítása vagy bővítése egyszerűbb a különböző rétegek elkülönülése miatt.

Observer (Megfigyelő) Minta:

Az Observer minta olyan minta, amely a változások figyelésére és azokra való reagálásra szolgál.

Komponensek:

- Alany (Subject): Az objektum, amelynek változásait megfigyelik.
- Megfigyelő (Observer): Az objektum, amely reagál az alany változásaira.

Működés:

1. Az alany regisztrálja a megfigyelőket.
2. Amikor az alany változik, értesíti a regisztrált megfigyelőket.
3. A megfigyelők reagálnak a változásokra, végrehajtva a szükséges műveleteket.

Előnyök:

- Rugalmas és laza kapcsolat az objektumok között.
- Az egyik objektum állapotváltozása azonnali értesítést jelent a többi objektumnak.

Factory (Gyár) Minta:

A Factory minta egy tervezési elv, amely szétválasztja az objektumok létrehozását a használatuktól.

Komponensek:

- Absztrakt Gyár (Abstract Factory): Definiálja az interfészt a konkrét gyár osztályok számára.
- Konkrét Gyár (Concrete Factory): Implementálja az absztrakt gyár interfészt és létrehozza a konkrét objektumokat.
- Absztrakt Termék (Abstract Product): Definiálja az interfészt a konkrét termék osztályok számára.
- Konkrét Termék (Concrete Product): Implementálja az absztrakt termék interfészt.

Működés:

1. A kliens kéri az absztrakt gyárt, hogy hozzon létre egy objektumot.
2. Az absztrakt gyár kiválasztja a megfelelő konkrét gyárat.
3. A konkrét gyár létrehozza a konkrét objektumot, és visszaadja azt a kliensnek.

Előnyök:

- Az objektumok létrehozása és inicializálása elkülönül a kliens kódtól.
- Különféle típusú objektumok létrehozására és inicializálására szolgál, anélkül, hogy a kliensnek részletekbe menően kellene ismernie azokat.

Builder (Építő) Minta:

A Builder minta egy tervezési elv, amely segíti az objektumok létrehozásának folyamatát összetett és összetett struktúrájú objektumok esetén.

Komponensek:

- Építő (Builder): Az interfész az objektum létrehozásának lépéseinek definiálásához.
- Konkrét Építő (Concrete Builder): Implementálja az építő interfészt és definiálja az objektumot.

- Termék (Product): Az elkészült objektumot reprezentálja.

Működés:

1. Az Igazgató értesíti az Építőt az objektum létrehozásáról.
2. Az Építő végrehajtja az objektum létrehozásának lépéseit.
3. Az Igazgató visszakapja az elkészült objektumot.

Előnyök:

- Megkönnyíti az objektumok különböző konfigurációinak és opcióinak alkalmazását.
- Lehetővé teszi az objektumok fokozatos és részletekbe menő felépítését.

Singleton Minta:

A Singleton minta egy tervezési elv, amely garantálja, hogy egy adott osztályból csak egy példány létezzen.

Komponens:

- Singleton: Az osztály, amelyből csak egy példány létezhet.

Működés:

- A Singleton osztályból csak egy példány van jelen az alkalmazásban.
- Globális hozzáférést biztosít az egyetlen példányhoz.

Előnyök:

- iztosítja, hogy egy adott osztályból csak egy példány létezzen.
- Globális hozzáférést biztosít az egyetlen példányhoz.

Adapter (Átalakító) Minta:

Az Adapter tervezési minta célja egy meglévő osztály interfészének átalakítása egy másik interfészre úgy, hogy a két interfész kompatibilis legyen, és az egyik osztály használható legyen a másik helyett. Ez segíti az együttműködést olyan osztályok között, amelyek különböző interfészekkel rendelkeznek.

Az Adapter minta egy strukturális tervezési minta, amely lehetővé teszi egy meglévő osztály használatát, anélkül hogy megváltoztatnánk az annak interfészét.

Komponensek:

- Célosztály (Target): A kívánt interfészt definiálja, amit az Adapter célba kíván hozni.
- Adaptált Osztály (Adaptee): A meglévő osztály, amelyet az Adapter használni fog.
- Adapter: Implementálja a Célosztály interfészt és tartalmazza az Adaptált Osztályt.

Működés:

1. A Célosztály definiálja az interfészt, amit a kliens használni szeretne.
2. Az Adaptee egy meglévő osztály, amelynek interfésze nem kompatibilis a Célosztály interfészével.
3. Az Adapter tartalmazza az Adaptált Osztályt és implementálja a Célosztály interfészét, átalakítva az Adaptee interfészét a kívánt formába.

4. A kliens a Célosztályt használja, anélkül hogy tudna az Adapter és az Adaptált Osztály létezéséről.

Előnyök:

- Rugalmasság: Az Adapter lehetővé teszi a meglévő osztályok használatát anélkül, hogy meg kellene változtatni azok kódját.
- Újrafelhasználhatóság: Az Adapter régi osztályokat hoz el a rendszerbe, újrafelhasználva azokat a Célosztály számára.
- Közvetítés: Az Adapter lehetőséget biztosít az Adaptált Osztály funkcióinak közvetítésére a kliens felé.

Command (Parancs) Minta:

A Command egy viselkedési tervezési minta, amely a kérést egy önálló objektummá alakítja, amely a kéréssel kapcsolatos összes információt tartalmazza.

Ez az átalakítás lehetővé teszi a kérések átadását metódusargumentumként, a kérés végrehajtásának késleltetését vagy sorba állítását, valamint a visszavonható műveletek támogatását.

A minta egy adott metódushívást önálló objektummá alakíthat. Ez a változás sok érdekes felhasználási lehetőséget nyit meg:
parancsokat adhatunk át metódusargumentumként, tárolhatjuk őket más objektumokon belül, futásidőben kapcsolhatjuk a kapcsolt parancsokat, stb.

Komponensek:

- Parancs (Command): Definiálja a kérés vagy operáció közös interfészét.
- Konkrét Parancs (Concrete Command): Implementálja a Parancs interfészt és tartalmazza az operációt és a hozzá tartozó paramétereket.
- Parancs Kibocsátó (Invoker): Meghívja a Parancsokat anélkül, hogy ismerné azok konkrét osztályait.
- Parancs Vevő (Receiver): Ténylegesen végrehajtja a kéréshez tartozó műveletet.
- Kliens: Konfigurálja és a Parancsokat kibocsátja.

Működés:

1. A Kliens létrehozza a konkrét Parancsokat és hozzárendeli nekik a megfelelő Vevőket.
2. A Parancsokat átadja a Kibocsátónak.
3. A Kibocsátó hívja a Parancsokat anélkül, hogy ismerné a konkrét Parancsokat.
4. A Parancs végrehajtja a műveletet a hozzárendelt Vevőn.

Előnyök:

- Rugalmasság: A Kliens nem ismeri a konkrét Parancsokat vagy Vevőket, ami lehetővé teszi könnyű cseréjüket.
- Sorba állíthatóság: A Parancsokat el lehet tárolni és később végrehajtani.
- Visszavonhatóság: A Parancsok visszavonhatók, mivel a teljes kérés információját tartalmazzák.

A tervezési minták alkalmazása segít a kód strukturálttá és karbantarthatóvá tételében az OO programozásban. Az MVC minta kiválóan alkalmas az alkalmazások szétválasztására, míg az Observer, Factory, Builder és Singleton minták további eszközöket nyújtanak a rugalmas és hatékony kód kialakításához. A megfelelő minták kiválasztása és alkalmazása kulcsfontosságú az alkalmazások tervezése során.