Tervezési minták az objektumorientált programozásban

Bevezetés

A tervezési minták olyan bevett megoldások, amelyeket a szoftverfejlesztők újra és újra használnak az objektumorientált programozásban. Ezek sablonok, amelyeket konkrét helyzetekhez alkalmazkodva használunk. Segítenek a kód szervezésében, és karbantarthatóbbá teszik az alkalmazásokat. A mintákat három csoportba szokás besorolni: kreációs (objektumok létrehozása), szerkezeti (objektumok összekapcsolódása) és viselkedési (objektumok közötti kommunikáció).

MVC (Model-View-Controller) minta

Az MVC az alkalmazások felépítésének egy elterjedt módja. Az alkalmazást három független rétegre bontjuk: a Model kezeli az adatokat és üzleti logikát, a View az adatok megjelenítéséért felel (weboldal, mobilfelület, asztali app), a Controller pedig a felhasználó eseményeit (kattintás, gombnyomás) fogadja és továbbítja a Model-nek feldolgozásra. Az előnye, hogy az egyes rétegek függetlenek: a megjelenítés módosítása nem érinti az adatkezelést. Szinte minden modern alkalmazás ezt használja.

Singleton minta

A Singleton biztosítja, hogy egy osztályból az egész program futása során csak egy példány létezzen. Például a naplózó (logger) modulnak egy példánya elég, az adatbázis-kapcsolatok pedig gazdaságosabbak egy példányban. Megvalósítása: az osztály konstruktorát priváttá tesszük, és egy statikus getInstance() metódus biztosít hozzáférést az egyetlen példányhoz. Első híváskor létrehozza az objektumot, majd azt gyorsítótárazva adja vissza.

Facade minta

A Facade minta egy összetett alrendszerhez egyszerű interfészt biztosít. Például egy számítógép indítása több lépésből áll (processzor, RAM, merevlemez inicializálása). A Facade egy "kapuőr" osztály, amely ismeri az összes bonyolult alrendszert és egyszerű metódusokat kínál, például startComputer(). A kliens-kód így csak ezt az egy metódust hívja meg, nem kell tudnia a belső bonyolultságról.

Command minta

A Command minta a műveleti kéréseket objektumként kezeli. Hasznos, amikor vissza szeretnénk tudni vonni (undo) a műveletek vagy azokat naplózni. Megvalósítása: definiálunk egy Command interfészt execute() metódussal, azután konkrét parancsokhoz osztályokat készítünk (CopyCommand, DeleteCommand). Az Invoker ezt a Command objektumot hívja meg. Az előnye: sorba rendezhetjük a parancsokat, visszavonhatjuk őket, és könnyedén naplózhatjuk a műveletek történetét.

State minta

A State minta akkor hasznos, amikor egy objektum viselkedése nagyban megváltozik az állapotától függően. Gyakorlati példa: okostelefon. Ha zárva van, gombnyomásra csak unlock képernyő jön. Ha nyitva van, a gombok alkalmazásokat indítanak. Ha az akkumulátor alacsony, csak töltési figyelmeztetés jelenik meg. Megoldás: definiálunk egy State interfészt, minden állapothoz osztályt készítünk (LockedState, UnlockedState, LowBatteryState), és az objektum az aktuális State objektumára delegálja a kéréseket. Ez sokkal tisztább, mint rengeteg if-else utasítás.

Összefoglalás

A tervezési minták garantálják, hogy a kód könnyebben érthető, karbantartható és bővíthető lesz. Az ilyen jól bevált megoldások követése megtérül: egy jól szervezett alkalmazás mellett sokkal könnyebb fejleszteni. Ezért szinte nélkülözhetetlenek az objektumorientált szoftverfejlesztésben.