Tervezési Minták az Objektumorientált Programozásban

Barkaszi Tamás

Iterator Minta

Az Iterator minta egy viselkedési tervezési minta, amely lehetővé teszi egy gyűjtemény, legyen az lista, verem vagy fa, elemeinek bejárását anélkül, hogy felfednénk a belső megvalósítást. A gyűjtemények az adattípusok leggyakoribb formái a programozásban, és minden gyűjteménynek szüksége van a hozzáféréshez szükséges módszerekre. Az iterátor lehetőséget ad arra, hogy különböző bejárási algoritmusokat, például mélységi vagy szélességi bejárást, különböző iterator osztályok létrehozásával implementáljunk.

Implementációja a következő lépésekből áll:

Az iterátor interfész deklarálása, amely legalább egy metodológiát tartalmaz.

Gyűjtemény interfész létrehozása, amely lehetővé teszi az iterátorok beszerzését.

Konkrét iterátor osztályok implementálása a gyűjteményekhez.

A gyűjtemény interfész beépítése a gyűjtemény osztályokba.

A kliens kódjának módosítása, hogy az életciklust teljesen az iterátorok kezeljék.

Előnyök:

Tiszta, karbantartható kliens kód.

Új gyűjtemények és iterátorok könnyű bevezetése.

Hátrányok:

Lehetséges túlzott bonyolítás egyszerű gyűjtemények esetén.

Teljesítménybeli csökkenés, amely a közvetlen hozzáférés helyett az iterátorok használata miatt léphet fel.

Command Minta

A Command minta egy viselkedési tervezési minta, amely egy kérés objektummá alakítását segíti elő, így lehetővé vált, hogy a kéréseket paraméterként továbbítsuk, késleltessük vagy sorba állítsuk. Ez a megközelítés megoldja a GUI (Grafikus Felhasználói Felület) és az üzleti logika közötti kommunikáció bonyolultságait.

Implementációja a következő lépésekből áll:

Command interfész létrehozása a végrehajtási metódussal.

Különböző, a Command interfészt megvalósító parancsok definiálása.

Kérő (invoker) osztályok létrehozása, amelyek a parancsokat tárolják.

A kliens inicializálja az összes komponenst a megfelelő sorrendben.

Előnyök:

Tiszta felelősségek a kód szervezésében.

Undo/Redo (visszavonás/újra végrehajtás) funkcionalitás megvalósítása.

Hátrányok:

A kód bonyolultabbá válhat, mivel egy új réteg kerül bevezetésre.

Proxy Minta

A Proxy minta egy szerkezeti tervezési minta, amely lehetővé teszi, hogy egy másik objektum helyettesítésére szolgáló elemet biztosítsunk. Ezáltal a proxy kontrollálhatja az eredeti objektumhoz való hozzáférést.

Implementáció részei:

Proxy osztály létrehozása, amely hivatkozik a valódi szolgáltatási objektumra.

A proxy metódusok implementálása a dolog céljainak megfelelően.

Késleltetett inicializálás megvalósítása.

Előnyök:

Az eredeti szolgáltatás védelme.

Életciklus-kezelés.

Hátrányok:

A kód bonyolultsága növekedhet az új osztályok bevezetésével.

Prototype Minta

A Prototype minta lehetőséget biztosít arra, hogy meglévő objektumokat máshol klónozzunk, csökkentve a kód repetitív jellegét és a másolás bonyolultságát.

Implementáció lépései:

Klónozási interfész létrehozása.

Alternatív konstruktorok definiálása a prototípusok számára.

Klónozó metódusok implementálása.

Előnyök:

Megszünteti az osztályok közvetlen dependency-jét.

Egyszerűsíti az összetett objektumok létrehozását.

Hátrányok:

Nehézségek a körkörös hivatkozások kezelése során.

Bridge Minta

A Bridge minta célja, hogy az absztrakciókat és implementációkat különböző hierarchiákra válassza szét, ezáltal rugalmasabbá téve a tervezést és fejlesztést.

Implementálása:

Az ortogonális dimenziók azonosítása.

Az absztrakciós interfészek létrehozása.

Az implementációs osztályok létrehozása, hogy kapcsolódjanak az absztrakcióhoz.

Előnyök:

Platformfüggetlen alkalmazások létrehozása.

Nyitott/zárt elv alkalmazása.

Hátrányok:

A mintányosítás növelheti a kód komplexitását, mivel több osztályt igényel.

Abstract Factory Minta

Az Abstract Factory minta lehetővé teszi több összefüggő vagy függőleges termékcsalád létrehozását, anélkül, hogy meg kellene határoznia az objektumok konkrét osztályait. Ezáltal lehetővé teszi az alkalmazás számára, hogy különböző termékcsaládokat használjon egy platformon, és könnyen cserélhesse azokat.

Implementáció lépései:

Absztrakt gyár interfész, amely deklarálja az összes szükséges termék létrehozásához szükséges metódust.

Konkrét gyárak, amelyek implementálják az absztrakt gyár interfészt.

Absztrakt termék interfészek, amelyek leírják a termékek közös viselkedését.

Konkrét termékek, amelyek megvalósítják az absztrakt termék interfészeket.

Előnyök:

Az alkalmazás rugalmasan bővíthető új termékcsaládokkal.

Elrejti a termékek konkrét megvalósításait a kliens elől.

Hátrányok:

A kialakítás bonyolultabbá válhat, mivel több részből áll és nagyobb mennyiségű kód szükséges.

Singleton Minta

A Singleton minta biztosítja, hogy egy osztályból csak egy példány létezzen, és globális hozzáférést biztosít annak példányához. Ezt a mintát gyakran használják olyan helyzetekben, ahol egy központi erőforrást kell kezelni, mint például konfigurációs beállítások vagy naplózás.

Implementáció lépései:

Privát konstruktor létrehozása, amely megakadályozza az osztály példányosítását kívülről.

Statikus metódus, amely visszaadja az osztály egyetlen példányát.

Opcionálisan megvalósítható szálbiztosítás annak érdekében, hogy az egyetlen példány egymás mellett futó szálakból is elérhető legyen.

Előnyök:

Csökkenti az erőforrások fogyasztását azáltal, hogy egy példányt használ.

Globális hozzáférést biztosít az adott objektumhoz.

Hátrányok:

Nehézkessé válhat a tesztelés, mivel a Singleton minta globális állapotot teremt.

A kód kevésbé lesz rugalmas, mivel a Singleton használata megnehezíti az osztályok függőségeinek kezelését.

State Minta

A State minta lehetővé teszi egy objektum számára, hogy a viselkedését megváltoztassa, amikor a belső állapota megváltozik. Ezzel a megoldással az állapot-specifikus viselkedési kód elválaszható, így a kód tisztább és karbantarthatóbb lesz.

Implementáció lépései:

Absztrakt állapot interfész létrehozása, amely deklarálja a viselkedési metódusokat.

Konkrét állapotok megvalósítása, amelyek implementálják az állapot interfészt.

A kontextus osztály, amely fenntartja az aktuális állapotot és delegálja a metódusokat az aktuális állapot objektumnak.

Előnyök:

Könnyen bővíthető új állapotokkal.

Az állapot-specifikus hatékonyság növelése.

Hátrányok:

A több állapot komplexitást adhat hozzá a kódhoz, mivel minden állapothoz külön osztály szükséges.