Programtervezési minták:

Létrehozási minták

Ezek a minták az objektumok létrehozásának módját optimalizálják, hogy a kód rugalmasabb és könnyebben karbantartható legyen.

Factory Method

Egy metódust biztosít az objektumok létrehozásához, de az alosztályokra bízza a konkrét típus meghatározását.

Abstract Factory

Egy interfészt definiál a kapcsolódó vagy egymással kompatibilis objektumok családjainak létrehozásához anélkül, hogy megadná azok konkrét osztályait.

Builder

Komplex objektumok fokozatos, lépésenkénti felépítését teszi lehetővé, miközben elrejti az objektum létrehozásának részleteit.

Prototype

Egy meglévő objektum klónozását biztosítja, ahelyett, hogy új példányt hozna létre.

Singleton

Biztosítja, hogy egy osztályból csak egyetlen példány létezhessen, és globális hozzáférési pontot biztosít ehhez.

Szerkezeti minták

Az osztályok és objektumok kapcsolatait optimalizálják, hogy a rendszer modulárisabb és hatékonyabb legyen.

Adapter

Egy meglévő osztály interfészét másik interfésszé alakítja, hogy a kliens kód használhassa.

Bridge

Különválasztja az absztrakciót és annak implementációját, hogy mindkettő önállóan módosítható legyen.

Composite

Lehetővé teszi az objektumok hierarchiájának kezelését úgy, hogy az egyes és összetett objektumokat ugyanúgy kezelhetjük.

Decorator

Dinamukusan hozzáad új funkciókat egy objektumhoz anélkül, hogy megváltoztatnánk az osztályát.

Facade

Egyszerűsített interfészt nyújt egy komplex alrendszerhez.

Flyweight

Több hasonló objektum esetén minimalizálja a memóriahasználatot az állapot megosztásával.

Proxy

Egy másik objektum helyettesítésére vagy elérésének ellenőrzésére szolgál.

Viselkedési minták

Az objektumok közötti kommunikációt és az algoritmusok elosztását segítik.

Chain of Responsibility

Egymást követő objektumok láncolatán továbbít egy kérést, amíg valamelyik nem kezeli azt.

Command

Egy műveletet parancsként csomagol, amelyet később végre lehet hajtani, naplózni vagy visszavonni.

Iterator

Lehetővé teszi, hogy egy kollekció elemein végighaladjunk anélkül, hogy feltárnánk annak belső szerkezetét.

Mediator

Centralizálja az objektumok közötti kommunikációt egy mediátoron keresztül.

Memento

Egy objektum belső állapotának mentését és visszaállítását teszi lehetővé anélkül, hogy feltárná annak belső részleteit.

Observer

Értesíti a megfigyelő objektumokat, ha az alany állapota megváltozik.

State

Lehetővé teszi, hogy egy objektum viselkedése dinamikusan változzon az állapota alapján.

Strategy

Különböző algoritmusokat biztosít egy családba csoportosítva, amelyek dinamikusan cserélhetők.

Template Method

Egy algoritmus vázát definiálja, miközben néhány lépését az alosztályokra bízza.

Visitor

Lehetővé teszi, hogy új műveleteket adjunk hozzá egy objektumstruktúrához anélkül, hogy annak osztályait megváltoztatnánk.

Proxy minta

A Proxy minta egy másik objektum helyettesítésére vagy annak elérésének ellenőrzésére szolgál. Egy Proxy osztály ugyanazt az interfészt valósítja meg, mint az általa képviselt objektum, így a kliens számára átláthatatlan, hogy valójában egy közvetítő réteget használ.

Mikor érdemes használni a Proxy mintát?

- Ha szeretnéd késleltetni egy erőforrás-igényes objektum létrehozását (pl. nagy adatbázis-hozzáférések vagy komplex fájlkezelés).
- Ha hozzáférés-szabályozást akarsz bevezetni (pl. jogosultságok ellenőrzése).
- Ha naplózni akarod az objektumokhoz való hozzáféréseket vagy statisztikát gyűjtesz.
- Ha távoli objektumokat szeretnél elérni (pl. hálózaton keresztül).

Típusai:

1. **Virtuális Proxy**: Az erőforrás-igényes objektum létrehozását késlelteti, amíg ténylegesen szükség van rá.
2. **Távoli Proxy**: Hálózati kapcsolatot absztrahál, hogy egy távoli szerveren lévő objektumot helyileg lehessen kezelni.
3. **Védelmi Proxy**: Hozzáférés-szabályozást biztosít.
4. **Gyűjtő Proxy**: Több kérést egyesít, mielőtt azokat az eredeti objektumnak továbbítja.

Előnyök:

- Csökkenti az erőforrás-felhasználást azáltal, hogy csak akkor tölti be az objektumot, amikor szükséges.
- Kliens oldali hozzáférést egyszerűsíti, miközben rejtve marad a valós objektum komplexitása.
- Biztonsági réteget vagy hozzáférés-szabályozást biztosít.

Hátrányok:

- Növelheti a kód komplexitását a közvetítő réteg miatt.
- Bizonyos esetekben teljesítményproblémát okozhat, például ha a Proxy túl sok ellenőrzést végez.

Observer minta

Az Observer minta egy viselkedési programtervezési minta, amely lehetővé teszi, hogy egy objektum értesítse a hozzá kapcsolódó más objektumokat a saját állapotának megváltozásáról. Ez a minta a publish-subscribe mechanizmus elvét használja, ahol az alany az információk "közlője", míg a megfigyelők az "előfizetők".

Mikor érdemes használni az Observer mintát?

Az Observer minta hasznos, ha:

1. Egy objektum állapotának változása más objektumokra is hatással van, de nem szeretnéd, hogy ezek az objektumok szorosan össze legyenek kötve.
2. Dinamikusan szeretnéd kezelni az értesítendő objektumokat, vagyis lehetőséged legyen megfigyelőket hozzáadni vagy eltávolítani futásidőben.

Observer minta résztvevői

1. **Subject**: Az objektum, amely eseményeket generál, és fenntartja a megfigyelők listáját.
2. **Observer**: Azok az objektumok, amelyek reagálnak az alany állapotváltozásaira.
3. **ConcreteSubject**: Az alany konkrét implementációja.
4. **ConcreteObserver**: A megfigyelők konkrét implementációi.

Előnyök:

- **Laza csatolás**: Az alany nem tud semmit a megfigyelőkről, csak azt, hogy értesíteni kell őket.
- Könnyen bővíthető: Új megfigyelőket adhatsz hozzá az alany kódjának módosítása nélkül.
- **Reaktív programozás alapja**: A minta alapvetően aszinkron értesítéseket biztosít.

Hátrányok:

- Ha túl sok megfigyelő van, az értesítések kezelése jelentős erőforrást igényelhet.
- Az értesítések sorrendje nem garantált.
- A ciklikus függőségek elkerülése érdekében óvatos tervezést igényel.

Factory Method minta

A Factory Method egy létrehozási programtervezési minta, amely egy közös interfészen keresztül biztosítja az objektumok létrehozását, miközben a konkrét osztályok példányosítását a gyermekosztályokra bízza. Ez lehetővé teszi, hogy a kódot rugalmasabbá és bővíthetőbbé tegyük azáltal, hogy elválasztjuk az objektumok létrehozásának folyamatát a felhasználásuktól.

Mikor érdemes használni a Factory Method mintát?

- Amikor az osztályok nem tudják előre, milyen típusú objektumokat kell létrehozniuk.
- Amikor azt szeretnéd, hogy az objektumok létrehozását alosztályokra bízd, miközben a kliens kódot függetlenül tartod a konkrét implementációktól.
- Amikor a rendszer különböző konfigurációk szerint hoz létre objektumokat, és ez a folyamat dinamikusan változhat.

Factory Method minta szerkezete

1. **Creator**: Tartalmazza a **factoryMethod()** absztrakt metódust, amelyet az alosztályok implementálnak az objektumok létrehozására.
2. **ConcreteCreator**: Implementálja a gyártó metódust, és konkrét objektumokat hoz létre.
3. **Product**: Egy interfész vagy absztrakt osztály, amelyet az összes konkrét termék implementál.
4. **ConcreteProduct**: A termék interfész konkrét implementációi.

Előnyök:

- A kód rugalmasabb és bővíthetőbb lesz, mivel az objektumok konkrét példányosítását elrejti.
- Csökkenti a kód ismétlődését és a karbantarthatósági problémákat.
- Az osztályok függetlenek a konkrét termékek implementációjától.

Hátrányok:

- Az osztályhierarchia bonyolultabbá válhat a gyártó és termék osztályok miatt.
- Több kódsor szükséges, mint ha egyszerűen konkrét objektumokat hoznánk létre.