Design Patterns

A **Design Patterns** (tervezési minták) olyan bevált megoldások gyűjteménye, amelyeket az objektumorientált szoftvertervezés során alkalmaznak az ismétlődő problémák megoldására. Ezek nem konkrét kódok, hanem magas szintű, absztrakt leírások arról, hogyan lehet megoldani egy adott problémát, figyelembe véve a szoftver struktúráját, rugalmasságát és fenntarthatóságát.

A Design Patterns a programozási gyakorlatokban használatos legjobb megoldások, amelyek segítenek:

- **Újrafelhasználható kódot** írni.
- **Könnyebben érthető és karbantartható programokat** létrehozni.
- **Kommunikációt javítani** a fejlesztők között azáltal, hogy közös nyelvet adnak.

Benefits of patterns

1. Újrafelhasználhatóság

- A tervezési minták újrahasználható megoldásokat kínálnak az ismétlődő problémákra.
- Csökkenti az időt és az erőfeszítést, amit a problémák újragondolására és megoldására fordítanánk.
- A Factory Method minta segítségével a különböző típusú objektumokat egy helyről lehet létrehozni, egységes módon.

2. Strukturált és tiszta kód

- A minták jól bevált szabályokat és elveket követnek, amelyek könnyen érthetővé és karbantarthatóvá teszik a kódot.
- Javítja a kód olvashatóságát, és lehetővé teszi más fejlesztők számára is a gyors megértést.
- Az Adapter minta segít különböző rendszerelemek összekapcsolásában anélkül, hogy módosítani kellene őket.

3. Skálázhatóság

- A minták segítségével a kód könnyebben bővíthető, új funkciók hozzáadása nem jár nagy átalakítással.
- A rendszer fejlődése során kevesebb fejlesztési munka szükséges.
- A Decorator minta lehetővé teszi, hogy új viselkedést adjunk hozzá egy objektumhoz anélkül, hogy módosítanánk az eredeti osztályt.

4. Kommunikációs eszköz

- A tervezési minták nevei (pl. Singleton, Observer, Proxy) egyfajta közös nyelvet biztosítanak a fejlesztők számára.
- Megkönnyíti a csapaton belüli kommunikációt és megértést, mivel a minta neve alapján mindenki tudja, milyen problémát old meg a kód.
- Ha valaki azt mondja, hogy a projektben "Observer mintát használunk", mindenki érti, hogy eseményfigyelésről van szó.

5. Hibák csökkentése

- A minták már bevált megoldások, ezért kevésbé valószínű, hogy hibás implementációhoz vezetnek.
- Csökkenti a tesztelési és hibakeresési időt.
- A Singleton biztosítja, hogy egy osztályból csak egy példány létezhessen, elkerülve a felesleges erőforrás-pazarlást.

6. Rugalmasság és jövőállóság

- A minták követése olyan rendszert eredményez, amely jól reagál a változásokra.
- Könnyebben hozzáadhatsz új funkciókat, módosíthatsz meglévőket anélkül, hogy az egész rendszert újra kellene írni.
- A Strategy minta lehetővé teszi, hogy az algoritmusokat könnyen cserélhessük vagy bővítsük.

7. Jobb időgazdálkodás

- A minták használata felgyorsítja a tervezési és fejlesztési folyamatot.
- Több időt fordíthatsz más feladatokra, például az üzleti logika kidolgozására vagy az alkalmazás felhasználói élményének fejlesztésére.

Classification

A **classification** (osztályozás) egy olyan adatbányászati és gépi tanulási módszer, amelynek célja, hogy egy adott bemeneti adatot előre meghatározott **kategóriákba** (osztályokba) soroljon. Ez egy **felügyelt tanulási** probléma, amely azt jelenti, hogy a modell tanítása során ismert címkékkel ellátott adatokkal dolgozunk. Az osztályozás (classification) az adatok kategorizálásának hatékony módja, amely segít automatizálni és optimalizálni a döntéshozatali folyamatokat. Különösen hasznos azokban az esetekben, amikor nagy mennyiségű adatot kell kezelni, és fontos a pontos, gyors eredmény.

Mi a célja az osztályozásnak?

- Az ismeretlen vagy új adatok kategorizálása.
- Az adatok jellemzői alapján döntések meghozatala.
- Olyan rendszerek létrehozása, amelyek automatikusan képesek az adatok osztályozására.

Hogyan működik az osztályozás?

1. **Adatgyűjtés:** Összegyűjtjük az adathalmazt, amely tartalmazza az attribútumokat (jellemzők) és az osztálycímkéket.
2. **Modell tanítása:** Az ismert címkékkel ellátott adathalmazon egy algoritmust (pl. döntési fa, logisztikus regresszió, neurális hálózat stb.) tanítunk.
3. **Tesztelés:** Ellenőrizzük, hogy a modell mennyire pontosan képes besorolni az új, ismeretlen adatokat.
4. **Használat:** A modellt éles környezetben használjuk az új adatok osztályozására.

Osztályozás típusai

1. **Bináris osztályozás:**
    - Két kategóriába sorolás (pl. spam vs. nem spam).
2. **Többosztályos osztályozás (Multiclass Classification):**
    - Több kategória (pl. állatfajok: kutya, macska, madár).
3. **Több címkés osztályozás (Multilabel Classification):**
    - Egy adathoz több címke is tartozhat (pl. egy cikk témái: "technológia" és "üzlet").

Osztályozás előnyei

1. **Automatizált döntéshozatal:** Gyorsan és pontosan osztályozhatunk nagy mennyiségű adatot.
2. **Pontosság javítása:** A gépi tanulási algoritmusokkal magas szintű osztályozási pontosság érhető el.
3. **Időmegtakarítás:** A manuális elemzés helyett a modellek gyorsan elvégzik a feladatot.
4. **Alkalmazhatóság:** Számos területen hasznos, például pénzügyi elemzésben, egészségügyben, marketingben és IT-ban.

Catalog of Design Patterns

A **design patterns** (tervezési minták) strukturált megoldások, amelyeket gyakran használt problémák megoldására dolgoztak ki az objektumorientált szoftvertervezésben.

1. Kreációs minták (Creational Patterns)

A kreációs minták az objektumok létrehozásának folyamatára összpontosítanak, hogy azokat rugalmasabbá és újrafelhasználhatóvá tegyék.

- **Singleton**
  Biztosítja, hogy egy osztályból csak egy példány létezzen, és globális hozzáférést biztosít hozzá.
  Példa: Konfigurációs menedzserek.
- **Factory Method**
  Egy interfészen keresztül hoz létre objektumokat, de a konkrét típusokat a leszármazott osztályok határozzák meg.
  Példa: Alkalmazások különböző UI-elemei (pl. gombok).
- **Abstract Factory**
  Kapcsolódó objektumcsaládok létrehozását teszi lehetővé anélkül, hogy megnevezné az egyes konkrét osztályokat.
  Példa: Cross-platform UI keretrendszerek.
- **Builder**
  Összetett objektumok lépésről lépésre történő létrehozását teszi lehetővé.
  Példa: Dokumentumok létrehozása különböző formátumokban.
- **Prototype**
  Az objektumokat másolással hozza létre az eredetiről (klónozás).
  Példa: Játékok karaktergenerálása.

2. Szerkezeti minták (Structural Patterns)

A szerkezeti minták az osztályok és objektumok közötti kapcsolatokra fókuszálnak, biztosítva, hogy a rendszer elemei együttműködőképesek legyenek.

- **Adapter**
  Egy meglévő interfészt más interfésszé alakít át, hogy kompatibilis legyen egy másik rendszerrel.
  Példa: Régi kódbázis illesztése új rendszerhez.
- **Bridge**
  Az absztrakciót és a megvalósítást szétválasztja, hogy külön-külön fejleszthetők legyenek.
  Példa: Eszközfüggetlen rajzolási rendszerek.
- **Composite**
  Lehetővé teszi, hogy egy objektumcsoportot ugyanolyan módon kezeljünk, mint egyetlen objektumot.
  Példa: Fájlok és mappák hierarchiája.
- **Decorator**
  Dinamikusan új viselkedéseket ad egy objektumhoz anélkül, hogy módosítaná az osztályt.
  Példa: UI-elemek vizuális díszítése.
- **Facade**
  Egy egyszerű interfészt biztosít egy bonyolult alrendszerhez.
  Példa: Komplex könyvtárak egyszerűsített API-ja.
- **Flyweight**
  Az objektumok közös részeinek megosztásával minimalizálja a memóriahasználatot.
  Példa: Karakterek megjelenítése szövegszerkesztőkben.
- **Proxy**
  Egy másik objektum helyettesítésére vagy vezérlésére szolgál.
  Példa: Hálózati hozzáférés, gyorsítótárazás.

3. Viselkedési minták (Behavioral Patterns)

A viselkedési minták az objektumok és osztályok közötti kommunikációt és együttműködést írják le.

- **Chain of Responsibility**
  A kérések továbbítását egy objektumlánc mentén szervezi meg, amíg valaki kezeli a kérést.
  Példa: Hiba- vagy eseménykezelő rendszerek.
- **Command**
  Egy műveletet objektummá csomagol, hogy később újra végrehajtható legyen.
  Példa: Visszavonás (undo) funkciók.

- **Interpreter**
  Egy nyelv szintaxisának és szemantikájának értelmezésére szolgál.
  Példa: Egyedi lekérdezési nyelvek implementálása.
- **Iterator**
  Egy gyűjtemény elemeinek sorrendbeli bejárását teszi lehetővé anélkül, hogy felfedné a belső szerkezetet.
  Példa: Lista vagy tömb iterálása.
- **Mediator**
  Az objektumok közötti közvetlen kapcsolat helyett egy közvetítő végzi a kommunikációt.
  Példa: Chat szobák.
- **Memento**
  Egy objektum állapotának pillanatképét menti, hogy később visszaállítható legyen.
  Példa: Játékok mentési rendszere.
- **Observer**
  Egy objektum változásaira más objektumokat értesít.
  Példa: Feliratkozási rendszerek (pl. eseményfigyelés).
- **State**
  Egy objektum viselkedését a belső állapota határozza meg.
  Példa: Játékok állapotkezelése (pl. "játszik", "szünetel", "vége").
- **Strategy**
  Algoritmusokat családként definiál, és azokat cserélhetővé teszi futásidőben.
  Példa: Fizetési rendszerek különböző módszerekkel.
- **Template Method**
  Egy művelet lépéseinek vázát definiálja, és a részleteket alosztályokra bízza.
  Példa: Egy dokumentum generálása különböző formátumokban.
- **Visitor**
  Új műveletek hozzáadását teszi lehetővé egy objektumstruktúrához anélkül, hogy módosítaná azt.
  Példa: Adatstruktúrák bővítése új feldolgozási szabályokkal.

History of patterns

A **tervezési minták** (Design Patterns) fogalma az építészetből származik, de a szoftverfejlesztésben vált igazán ismertté és széles körben használt megközelítéssé. A tervezési minták az építészeti elvekből fejlődtek ki, és mára a szoftverfejlesztés egyik legfontosabb eszközévé váltak. Az elmúlt évtizedekben a GoF munkája meghatározta az alapokat, amelyekre modern fejlesztési megközelítések és minták épülnek. A tervezési minták segítenek a fejlesztőknek jobb, tisztább és rugalmasabb rendszereket létrehozni.

Modern alkalmazások

- **Agilis fejlesztés és DevOps:** A tervezési minták ma is fontosak a gyors, rugalmas fejlesztési folyamatokban.
- **Cloud és elosztott rendszerek:** Új minták jelentek meg, például a mikroszolgáltatás-architektúrákhoz és a felhőalapú rendszerekhez (pl. Circuit Breaker, Retry Pattern).
- **Adatfeldolgozás és mesterséges intelligencia:** Minták alkalmazása az adatfolyamok kezelésére és a gépi tanulási modellek fejlesztésére.

Hatás a szoftverfejlesztésre

- A tervezési minták széles körben használt eszközzé váltak, mert:
    - **Újrafelhasználható megoldásokat** nyújtanak.
    - **Egységes nyelvet** biztosítanak a fejlesztők között.
    - **Javítják a kód minőségét** és karbantarthatóságát.
- A minták ma már az objektumorientált tervezés alappillérei, és részei a legtöbb fejlesztői oktatásnak és gyakorlatnak.

Criticism of Design Patterns

A **tervezési minták** bár elismert és széles körben használt eszközök a szoftverfejlesztésben, nem mentesek a kritikáktól.

A tervezési minták hasznos eszközök, de nem csodafegyverek. Használatuk előzetes megértést és körültekintő alkalmazást igényel. A kritikák arra figyelmeztetnek, hogy a mintákat nem szabad automatikusan minden helyzetre alkalmazni; gondosan mérlegelni kell, hogy adott esetben valóban indokolt-e használatuk. A megfelelő egyensúly megtalálása a hatékony fejlesztés kulcsa.