Tervezési minták objektum orientált programozásban

- A tervezési minták segítenek a fejlesztőknek a kódjukat rugalmasabbá és karbantarthatóbbá tenni. Minden mintának saját használati esete és előnyei vannak, a kiválasztás pedig függ a konkrét problémától, vagy követelményektől.

1. **Singleton:** Biztosítja, hogy egy adott osztályból csak egy példány létezzen, és egyetlen globális hozzáférést biztosít ehhez.

2. **Factory:** Elhalasztja egy osztály példányosítását az öröklődő osztályokra, így azok a létrehozás módját változtathatják, de a használati mód nem változik.

3. **Abstract Factory:** Biztosít egy interfészt a hasonló vagy függő objektumcsoportok létrehozásához anélkül, hogy pontosan megmondaná, melyik implementációt kell használni.

4. **Builder:** Segít az objektum létrehozásának összetett lépéseiben úgy, hogy ugyanazt a folyamatot alkalmazva különböző reprezentációkat hoz létre.

5. **Prototype:** Definiál egy objektumot, amely a prototípusként szolgál egy másik objektum példányosításához anélkül, hogy annak osztályát pontosan meghatározná.

6. **Adapter:** Engedélyezi egy osztály használatát egy másik osztállyal összekapcsolt különböző interfészen keresztül.

7. **Decorator:** Dinamikusan kiterjeszti vagy módosítja egy osztály működését anélkül, hogy az osztályt módosítaná.

8. **Proxy:** Biztosít egy helyettesítő vagy közvetítő objektumot, hogy kontrollálja egy másik objektum hozzáférését.

9. **Observer:** Meghatározza egy objektum függőségi láncolatát úgy, hogy bármelyik változás az egyik objektumot a többi objektumra értesíti.

10. **Strategy:** Definiál egy család egyesített algoritmusból álló algoritmusokat, és azokat úgy készíti el, hogy azok egymás helyettesíthetők legyenek.

11. **Command:** Egy kérés objektumként való reprezentálását teszi lehetővé, így lehetővé válik az ügyfelek számára, hogy kérés objektumokat paraméterezzék, sorolják fel, mentéssel és visszavonással dolgozzanak.

12. **State:** Egy objektum viselkedését változtatja meg, amikor az állapota megváltozik.

13. **Template:** Definiál egy algoritmust egy alaposabb részletezés nélkül, és hagyja, hogy azokat az alárendelt osztályok biztosítsák.

14. **Interpreter:** Egy nyelvi elemet reprezentáló nyelvi elemeket, és egy interpreter osztállyal értelmezi ezeket.

15. **Composite:** Objektumokat kompozit struktúrákba szervez, hogy rész-egész hierarchiákat reprezentáljon. Lehetővé teszi, hogy a kliensek egyes objektumokat és objektumösszetevőket egységesen kezeljenek.

16. **Flyweight:** A memóriahasználatot vagy a számítási kiadásokat minimalizálja, azáltal hogy a lehető legtöbbet megosztja a kapcsolódó objektumokkal; hatékony kezelést tesz lehetővé nagy számú hasonló objektum esetén.

17. **Chain of Responsibility:** Egy kérés átadása egy láncban lévő kezelők során. Amikor egy kérés érkezik, minden kezelő eldönti, hogy feldolgozza-e a kérést, vagy átadja a következő kezelőnek a láncban.

18. **Mediator:** Meghatároz egy objektumot, amely centralizálja a rendszer objektumai közötti kommunikációt. Csökkenti a kommunikáló objektumok közötti függőségeket.

19. **Memento:** Rögzíti és külsővé teszi egy objektum belső állapotát, hogy később ebben az állapotban lehessen visszaállítani az objektumot.

20. **Visitor:** Megtestesít egy műveletet, amit az objektumstruktúrák elemein kell végrehajtani. Lehetővé teszi, hogy új műveletet definiáljunk anélkül, hogy megváltoztatnánk azoknak az elemeknek az osztályait, amelyeken végrehajtjuk.

Tóth Gergő
LRTMQW

Források:
https://refactoring.guru/design-patterns/creational-patterns

https://okt.inf.szte.hu/prog1/gyakorlat/eloadas/TervezesiMintak/mintakAttekintese/