



kijelöljük az objektum részeit. Az objektum protected kulcsszóval deklarált része private elérési a kívülág számára, de ha saját osztályt származtatunk a védett elemekkel rendelkező osztályból, akkor public elérésűvé válnak az összetevők. Az osztályon belül a fenti kulcsszavakkal megjelölt részek teszőleges számban és sorrendben helyezkedhetnek el, de megkötés, hogy az adatok után a metódusok következnek.

**-Többrétegűség (polymorphism).** Egy adott metódus azonosítója közös lehet egy adott objektumhierarchián belül, ugyanakkor a hierarchia minden egyes objektumában a tevékenységeket végrehajtó metódus implementációja az adott objektumra nézve specifikus lehet (pl: virtuális metódusok).

29. Ismertesse a programtervezési mintákat általánosan, valamint mutassa be az osztályozásukat.

**Programtervezési minták (Design Pattern)**

Design Patterns, Újrahasznosítható osztályok: igazodnia kell a megoldandó problémához [] eléggé általánosnak kell lennie ahhoz, hogy később könnyen módosítható legyen  
Típusfeladatokra bevált megoldások, Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides (GoF - Gang of Four), **23** minta, **Minta:** „Egymással együttműködő objektumok és osztályok leírása, amely testreszabott formában valamilyen általános tervezési problémát old meg egy bizonyos összefüggésben.”, Azonosítja a résztvevő osztályokat és objektumokat, szerepüket és kapcsolataikat

**Tervezési minták:** Singleton (Egyke), Simple Factory, Adapter (Illesztő), Bridge (Híd), Composite (Összetétel), Observer (Megfigyelő), Strategy, Dependency injection, Template Method (Sablon), MVC, MVP, MVVM, Repository, Data Mapper, Facade,

30. Ismertesse az Egyke/Singleton tervezési mintát részletesen a C# nyelv használatával.

**Egyke – singleton, A minta neve és besorolása:** Egyke, objektum-létrehozási minta  
Cél: Egy osztályból csak egy példányt engedélyezni, és ehhez globális hozzáférési pontot megadni **Egyéb nevek:** Singleton, **Feladat:** Egyes osztályok esetén fontos, hogy pontosan egy példány legyen belőlük. Egy rendszerben több nyomtató is lehet, de nyomtatási sorból csak egyet lehet használni. Vagy csak egy fájlrendszer és csak egy ablakkezelő futhat. **Hogyan biztosíthatjuk, hogy egy osztályból csak egyetlen példány legyen, de azt könnyen el lehessen érni?** Ha globális változót hozunk létre, akkor az objektum könnyen elérhető lesz, de akkor akárhány példányt létrehozhatunk belőle. **Maga az osztály a felelős a példányok nyilvántartásáért.**

**C# megvalósítás:**

```
public class Egyke
{
    private static Egyke EgyediPeldany;
    public static Egyke Peldany()
    {
        if (EgyediPeldany == null)
            EgyediPeldany = new Egyke();
        return EgyediPeldany;
    }
    protected Egyke(){ ... }
}
```

31. Ismertesse az Adapter tervezési mintát részletesen C# nyelv használatával.

**Adapter – Illesztő**

A cél egy olyan új osztály/objektum létrehozása, ami a korábbi osztály felhasználásával megvalósít egy elvárt új interfészt

**Például:** a korábbi osztály rendelkezik **Vezetéknév** és **Utónév** tulajdonságokkal, de **Teljesnév** tulajdonsággal nem, és nekünk erre a tulajdonságra van szükségünk  
**Megvalósítási módok:** Öröklődés, Beágyazás, Kiegészítés, Bővítő metódus

32. Ismertesse a Simple factory tervezési mintát részletesen.

A *Simple Factory* pattern több lehetséges osztályból ad vissza egy példányt, a bemeneti paraméter(ek) függvényében. Általában ezen osztályoknak közös az őse és ugyanazokat a metódusokat tartalmazzák, de mindegyik máshogyan végzi el az adott feladatokat. Ebben a felállításban *X* az ösosztály, aminek van két leszármazottja, *XY* és *XZ*. Az *XFactory* osztály maga a factory, és a bemeneti paraméter függvényében eldönti hogy melyik leszármazott osztály (*XY* vagy *XZ*) egy példányát adja vissza. Ezt a *getClass* metódus végzi. Mivel mindkét leszármazott osztálynak ugyanazon metódusai vannak, ezért a programozó szempontjából lényegtelen melyiket adja vissza a factory, ugyanúgy lehet használni őket. A döntés, hogy melyik osztály példányosul majd, teljesen a factory felelőssége.

33. Ismertesse a Composite tervezési mintát részletesen.

Composite structure diagram

Megmutatja egy osztály belső szerkezetét és az egységek közötti együttműködési lehetőséget.

34. Ismertesse az Observer tervezési mintát részletesen.

A megfigyelő minta lehetővé teszi, hogy a rendszer egy része értesüljön arról, hogy a rendszer egy más részében bekövetkezik egy esemény. Ez 1:n függőségi kapcsolat az objektumok között. Ha a kiemelt objektum állapota változik, akkor a függő objektumok automatikusan értesülnek (és frissülnek).

Alkalmazási terület: ha egy objektumban bekövetkező változás esetén több más objektum változtatása szükséges. **Szerepkörök:**

**A Subject** a megfigyelés tárgya. Ebből egy van. Kell rendelkezzen egy olyan interfésszel, ami lehetővé teszi, hogy a megfigyelők futási időben rákapcsolódhassanak illetve lekapcsolódhassanak. Ezt biztosítják az Attach és a Detach metódusok.

**Az Observer** a megfigyelő. Ebből lehet több. Kell rendelkezzen egy olyan interfésszel, ami lehetővé teszi, hogy értesítést fogadjon a Subject-től. Ezt biztosítja az Update() metódus.

A gyakorlatban ez nem jelent mást, mint eseményvezérelt alkalmazás fejlesztését. A Subject eseményt tesz közzé, amire az Observer eseménykezelő metódusa feliratkozik. Az Observer mintát valósítja meg a klasszikus közzétévő-feliratkozó modell, amivel Vizualis programozás tárgy keretében találkoztunk.

35. Ismertesse a Strategy tervezési mintát részletesen.

Egy algoritmus családot definiál, befoglalja és cserélhetővé teszi az érintett algoritmusokat. Olyan esetekben ajánlott az alkalmazása, amikor egy alkalmazás egy bizonyos szolgáltatást igényel, és ez a szolgáltatás többféleképpen oldható meg. Az algoritmus kiválasztás történhet a felhasználó által vagy számítási hatékonyság, stb. alapján. Példa: különböző tömörítési eljárások alkalmazása. Mindegyik viselkedési mód/megoldást külön osztállyal (ConcreteStrategyA..C) valósítják meg.