**Progtech beadandó**

## Készítette: Szalóki Dávid, Seres Péter , Kerepesi Gergő

# Projektről röviden:

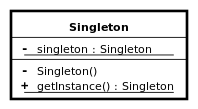
Szeretnénk készíteni egy Vatera alkalmazást, ahol lesz egy nagy Raktár. Amiből a Felhasználók tudnak válogatni a benne lévő Termekből. Az alkalmazásunk ezt hivatott megvalósítani.

Tervben : felhasználónak bevásárló kosár -> a rendeléshez

# Csomagok:

Terv: C# singleton + Diszítő tervezési minták

Singleton használata a raktár létrehozásához, csak egy raktár létezik, amiből lehet kivenni és pakolni. Új raktár létrehozását pedig megtiltjuk a Singleton tervezési minta alkalmazásával.



# Solid elvek:

**ISP (Interfészszegregációs-alapelv)** : A szolgáltatást nyújtó osztály fölé elhelyezünk egy interface-t hogy a kliens csak azokat a metódusokat láthassa amit valóban használ. ( Minden klienshez külön interface-t készítünk )

**Demeter törvénye** : Legkisebb tudás elve kimondja hogy egy osztály csak a közvetlen ismerőseit hivhatja

# Osztályok:

**Interfészek**: A különböző szinteken található interfészeket csak azon osztály implementálják, amiknek szükségük van a bennük lévő metódusokra.

**Absztrakt osztályok**: A legutolsó gyermek osztály kivételével, minden szülő absztrakt. Ennek célja, hogy a közös tulajdonságok összegyűjtve legyenek, így megspórolásra kerülnek a fölösleges ismétlések. Ennek lényege, hogy az azonos tulajdonsággal rendelkező dolgok kódjai csak egyszer legyenek megírva, viszont, ha a tárgyak különböznek (például a leírásaik), akkor azt a legalsó szinten lehessen kifejteni.

Az osztályokhoz kapcsolódóan felhasználtuk az **OOP alapelveket**

**-Egységbezárás (Encapulation) :** Az adattagokatés a hozzájuk kapcsolódó műveleteket egy egységbe zártuk (class)

**-Öröklődés (Inheritance) :** A gyermek osztály az ős minden mezőjét és metódusát megörökli

**-Többalakúság (Polymorphism) :** A tervezési minták szinte mindegyike ezen alapszik , maga a többalakúság az öröklődés következménye. Mivel a gyermek örökli az ős felületét igy megkapja azok típusait is, igy az objektum több alakban is használható

# Felhasznált tervezési minták:

**Singleton:** Olyan osztályok, amelyeket csak egyszer van szükségünk példányosítani mert csak a metódusokra van belőlük szükség. Nem hozható belőlük létre új példány PL: különböző storagek vagy factoryk. Kiegészítés : többszálú alkalmazás esetén le kell védeni a péndányosítást.

Felhasznált viselkedési minta:

**Megfigyelő-Observer:** Az observer egy olyan viselkedési minta, amely az egy-több kapcsol létrehozására alkalmas objektumok között. Ha az egyik objektumot megváltoztatjuk, az observer értesítést ad a tőle függő objektumoknak, és automatikusan frissíti őket.

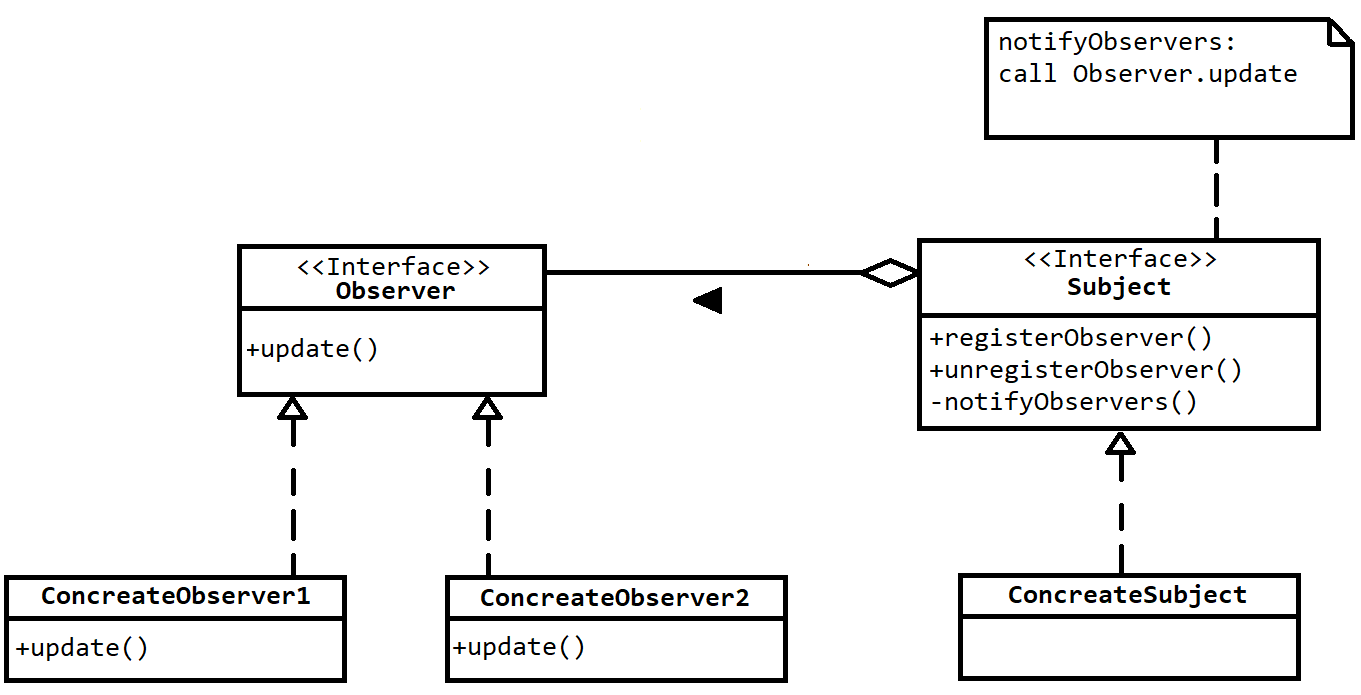
Egy olyan minta, amely az objektum (Subject) fenntartottainak listáját (Observer) tárolja, és egy metódus meghívásával automatikusan értesíti őket, ha egy állapot megváltozik.

Célja egy Modell-Nézet-Vezérlő, vagy röviden MVC létrehozása.

**Részei:**

* **Subject:** Interfészt biztosít a Megfigyelő objektumok csatolására és leválasztására.
* **Observer:** Frissítő interfész az értesítendő objektum/ok számára (update művelet).
* **ConcreteSubject:** A ConcreteSubject érdekes állapotokat tárol az Observer számára, és saját állapotváltozásairól értesíti őket.

**Példa:**



**Saját megoldás:** A mi esetünkben az ISubject lehetőséget biztosít csatolásra, illetve leválasztásra a megfigyelő objektumok számára, azaz member-ek számára. Konkrét tárgyunk a WishList, egy lista, ami tárolja az adott member-t, illetve egy item-et, ami éppen nincs raktáron. Ha az admin feltölti a raktárt az adott item-el, akkor a feltöltést követően az adott member értesítést kap a frissítésről.

## Felhasznált Tervezési Minta:

Díszítő/Dekorátor Tervezési Minta: A dekorátor tervezési minta megengedi nekünk, hogy dinamikusan adjunk új funkcionalitásokat egy már létező objektumhoz, anélkül, hogy annak struktúráját módosítanánk. Ez a tervezési minta, mint „csomagoló” fog viselkedni a már létező osztályunknak.

A minta lehetővé teszi számunkra egy objektum funkcionalitásának futási időben való módosítását.

Egy dekorátor egy olyan objektum, amely funkciókkal bővít egy másik objektumot.

**Előnyei a dekorátor mintának:**

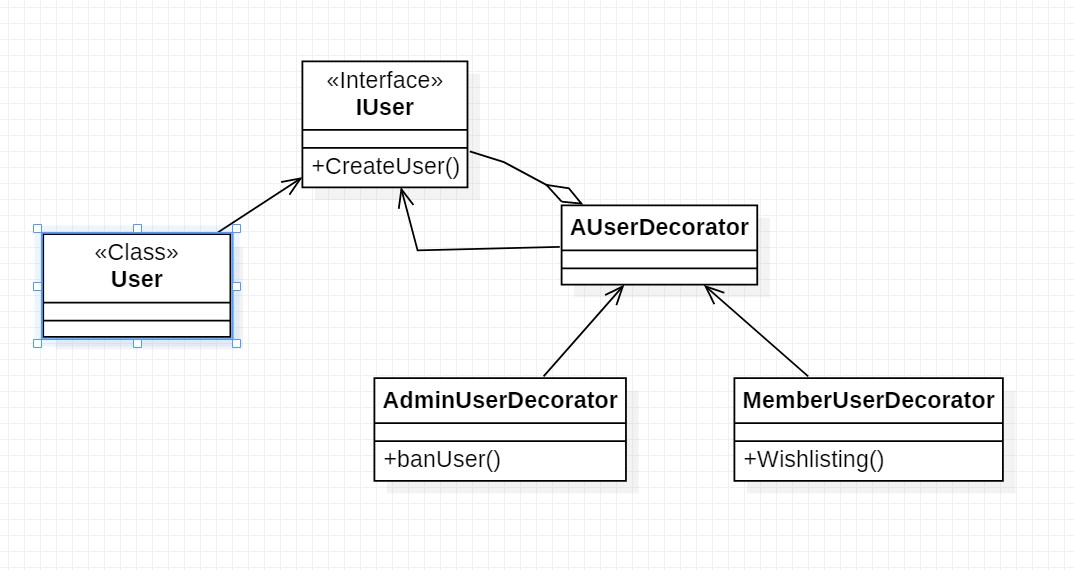
* Dinamikusan ad funkcionalitást létező objektumoknak
* Jó alternatíva az alosztályokra
* Flexibilis
* Támogatja a nyitva-zárt alapelvet

**Mikor érdemes használni:**

* Hagyaték rendszerekben
* Lezárt osztályoknál

**Részei:**

* Komponens (IUser)
* Konkrét Komponens (User)
* Dekorátor (AUserDecorator)
* Konkrét Dekorátor (AdminUserDecorator)



Esetünkben a felhasználóinkat díszítjük fel különböző operációs jogokkal. Kezdetben nem rendelkeznek semmilyen joggal, kizárólag a regisztrációhoz szükséges mezőkkel. Majd dekoráljuk a kívánságlistának használati jogával, illetve admin fiók esetében az ezzel járó jogkörökkel.