Template Method

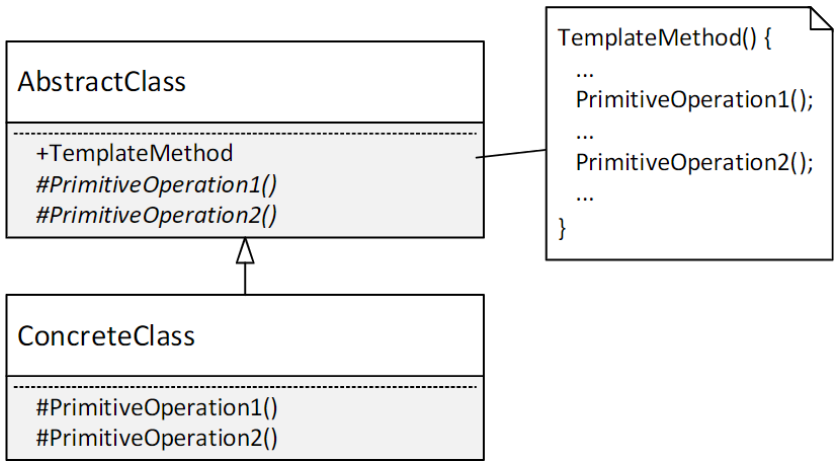
Jellemzők

- Egy műveleten belül algoritmus vázat definiál, és az algoritmus bizonyos lépéseinek implementálását a leszármazott osztályra bízza.
- A változatlan kódrészeket ősosztályba írjuk.
- A változó/kiterjeszthető részeket nem drótozzuk az ős műveleteibe, hanem *virtuális* és/vagy *absztrakt* függvényekre bízzuk.
- A leszármazott osztályban ezen függvények felülírásával adjuk meg a specifikus viselkedést.
- Lehetővé teszi, hogy az algoritmus/folyamat invariáns részeit egy helyen definiáljuk és a változó részeket a leszármazott osztályban adjuk meg.
- Elkerülhető a kódduplikáció (DRY-elv): a hierarchiában a közös kódrészeket a szülő osztályban egy helyen (*sablonmetódusban*) adjuk meg.

Összefoglalás

Van egy algoritmusod, amiből egyik-másik lépés megvalósítását többféleképpen kell megoldani, de a váz maga azonos.

Megoldás: Leszármazás és absztrakt metódusok.



Strategy

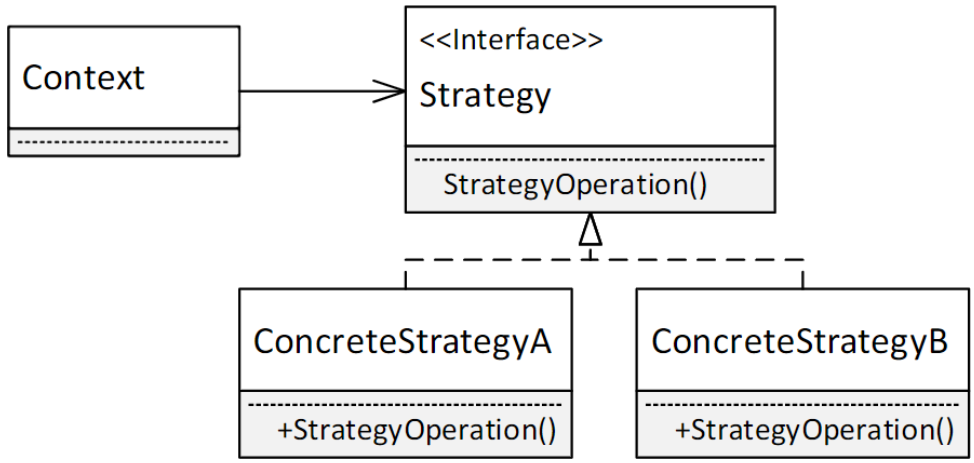
Jellemzők

- Célja az algoritmusok/viselkedések egy csoportján belül az egyes algoritmusok/ viselkedések egységbe zárása és egymással kicserélhetővé tétele.
- Az osztály viselkedésének minden olyan aspektusára/dimenziójára, melyet lecserélhetővé/bővíthetővé szeretnénk tenni, külön *strategy* hierarchiát vezetünk be.
- Az osztály külön (*interfész* típusú) hivatkozást tartalmaz minden viselkedés aspektusra/dimenzióra.
- Új viselkedés könnyen bevezethető, nem kell a meglévő kódot (lényegi helyen) változtatni.
- **Dependency Injection:** konstruktor, vagy esetleg metódus paraméterben adjuk át az implementációt az osztálynak, melyre az osztály interfészként hivatkozik az és azt kívülről kapja meg.

Összefoglalás

Van egy algoritmusod, ahol le akarsz cserélni lépéseket, de jellemzően két, vagy több egymástól független aspektus mentén is. A Template Method itt nem jó, mert az összes lehetséges kombinációban kellene leszármazott osztály.

Megoldás: Interfészek, minden aspektusnak egy.



Observer

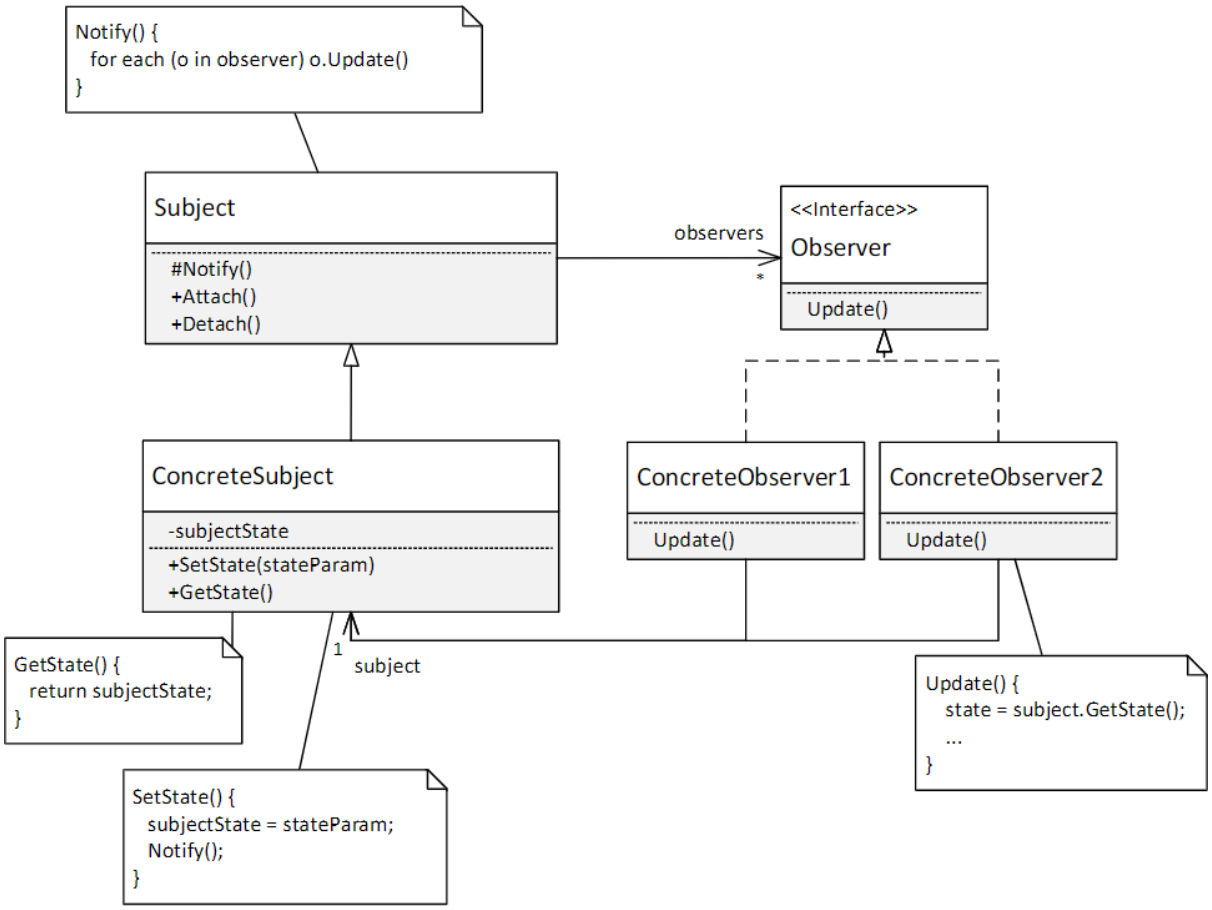
Jellemzők

- Lehetővé teszi, hogy egy objektum (*subject*) értesítést küldjön más objektumoknak (*observer*) az állapotának változásairól. Mindezt anélkül, hogy az alany függene a megfigyelők konkrét típusától.
- Az adatoknak egyetlen, közös „hiteles" forrása legyen: emeljük ki az adatokat és az azon értelmezett műveleteket egy osztályba, ez lesz a dokumentum (*subject*).
- A dokumentumhoz különböző nézeteket lehet beregisztrálni (*observer*).
- Ha valamelyik nézet megváltoztatja a dokumentum adatait, a dokumentum értesíti az összes beregisztrált nézetet a változásról.
- Az értesítés hatására a nézet lekérdezi a dokumentum állapotát és frissíti magát.
- A dokumentum csak egy közös *View* interfészen/ősosztályon keresztül tárolja a beregisztrált nézeteket (nem függ az egyes nézet típusoktól).
- Ez maga a *Document-View architektúra*.

Összefoglalás

Van valamid, ami változik és több olyan dolog, ami valahogyan reagál a változásokra, pl. egy kijelzőn megjelenik egy státusz. A figyelők feliratkoznak a megfigyeltre és az értesíti mindet.

Megoldás: Attach-Notify-Update.



Singleton

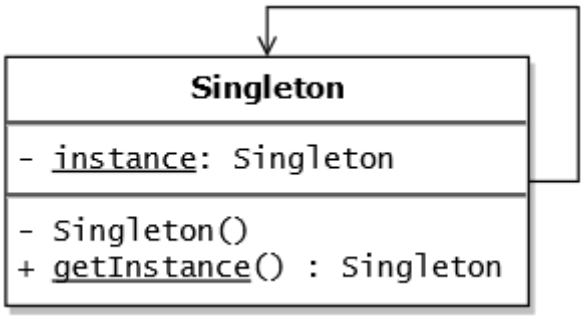
Jellemzők

- Biztosítja, hogy egy osztályból csak egy példányt lehessen létrehozni, és ehhez az egy példányhoz globális hozzáférést biztosít.
- Az egy példány kódban bárhonnan kényelmesen elérhető anélkül, hogy függvényparaméterben kellene átpasszolni.
- Az osztály felelőssége, hogy csak egy példányt lehessen belőle létrehozni: maga az osztály biztosít globális hozzáférést ehhez az egy példányhoz.
- Az egyetlen példányt maga az osztály tárolja egy statikus, pl. *Instance* nevű védett (private) változóban.
- Az egyetlen példányt a többi osztály egy statikus, pl. *GetInstance()* nevű metódussal éri el.
- Az osztály konstruktora védett (*private* vagy *protected*), így más osztályok nem tudják a *new* operátorral példányosítani. Ez garantálja, hogy más osztály nem tudja megkerülni a *GetInstance()* használatát, így további példányokat nem tud létrehozni.

Összefoglalás

Van egy információ, vagy szolgáltatás, amit bárhonnan (bármelyik osztályból) el akarsz érni és nem akarod összedrótozni mindegyikkel. Az információt tároló objektumból csak egy lehet, pl. egy központi szerver.

Megoldás: Rejtett konstruktor és statikus példánylekérés.



Abstract Factory

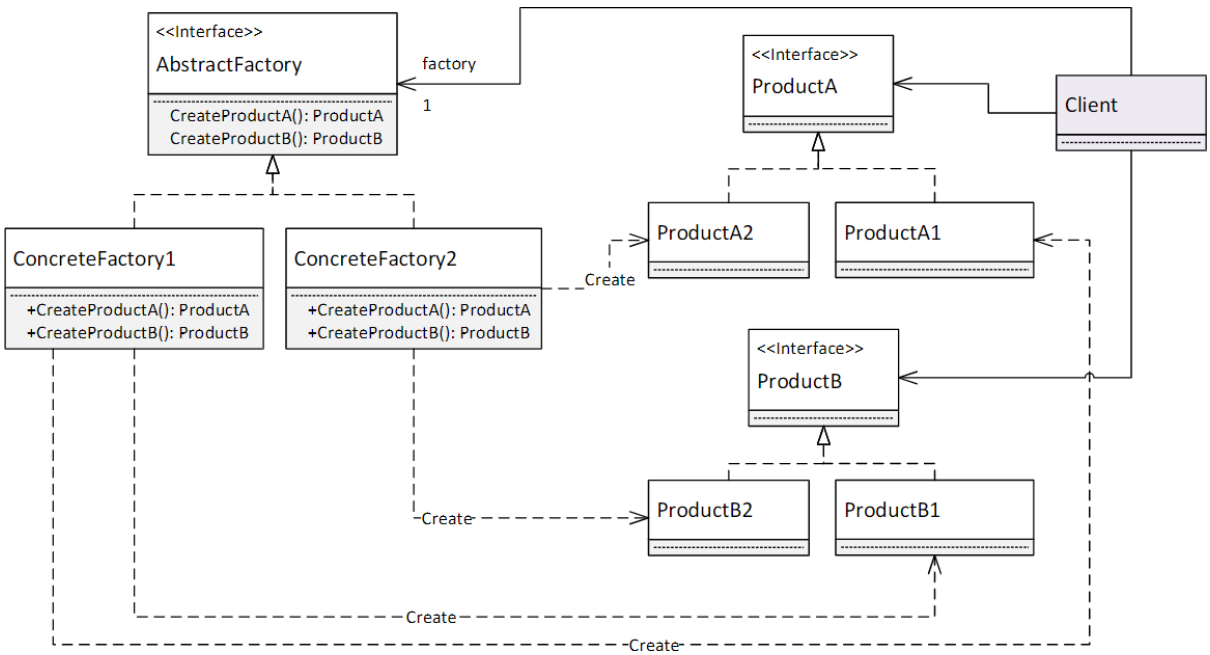
Jellemzők

- Interfészt biztosít ahhoz, hogy egymással összefüggő objektumok családjait hozzuk létre anélkül, hogy specifikálnánk a konkrét osztályaikat.
- Így az objektumok létrehozása egy interfészen keresztül történik, a kódunk nem fog függeni a létrehozott objektumok konkrét osztályától/típusától.
- Egységbe zárjuk az elemek létrehozását és más (*abstract factory*) objektumokra bízzuk.
- Az alkalmazásban csak egy interfészen keresztül hivatkozunk rájuk, így az implementációjuk kicserélhető.

Összefoglalás

Készítesz valamit, ami több részből áll és több gyártási módszered lehet. Fontos, hogy minden részt ugyanazzal a módszerrel kell gyártanod, nem keverheted őket, pl. Teslát és Trabantot gyártasz, nem kevered az alkatrészeiket.

Megoldás: Interfészek minden termékcsaládhoz, melyek konkrét típusokat állítanak elő és elszigetelik azokat egymástól.



Command

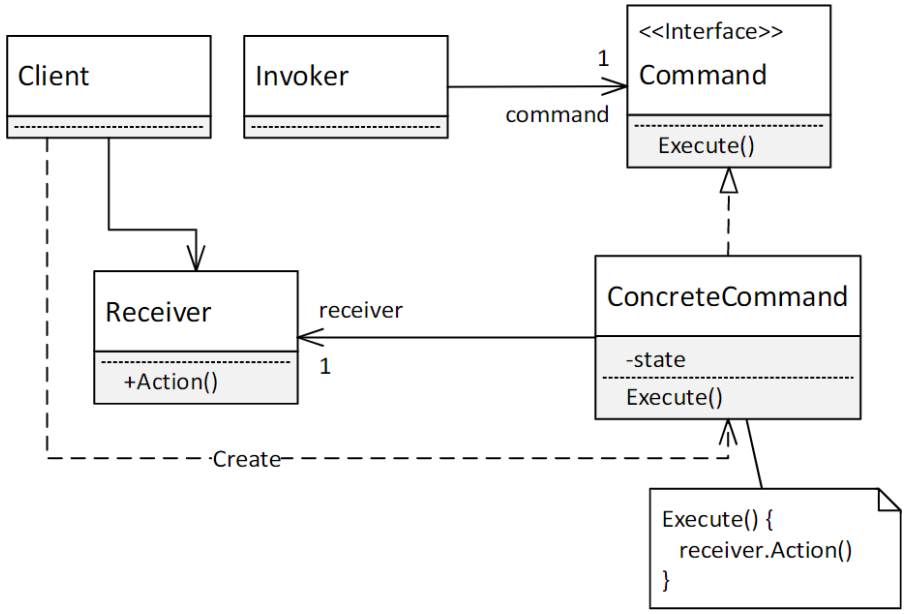
Jellemzők

- Egy kérés objektumként való egységbezárása.
- Egy kérés általában egy függvényhívásként jelenik meg a kódban. Ezzel szemben *Command* esetében a kérés objektumként jelenik meg.
- Ez lehetővé teszi a kliens különböző kérésekkel való felparaméterezését, a kérések sorba állítását, naplózását és visszavonását (*Undo*).
- Elválasztja a parancsot kiadó objektumot attól az objektumtól, amelyik tudja, hogyan kell lekezelni (adott *Command* implementáció).

Összefoglalás

Van több műveleted, amit "vezérszóra" akarsz végrehajtani. A műveletek paraméterezése (létrehozása) eltérő lehet, de a végrehajtásuk azonos módon kell, hogy történjen.

Megoldás: Interfészbe foglalás, amely egységesíti a végrehajtást.



Command Processor

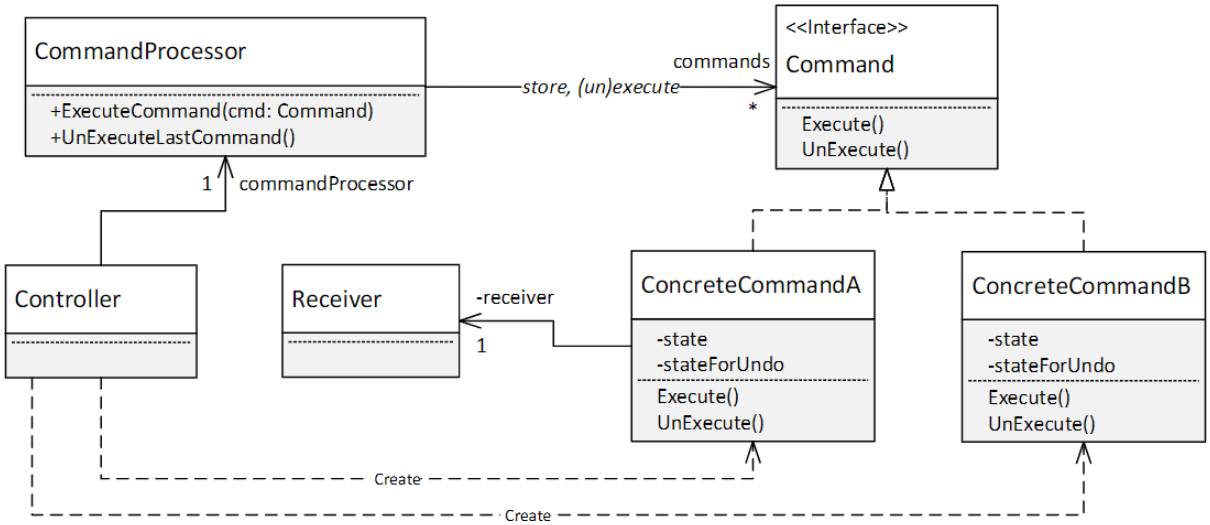
Jellemzők

- A *Command* minta egy változata.
- Beépítetten támogatja parancsok visszavonását (*Undo*)
- **Alapelvek:**
  - *Unexecute()* művelet bevezetése a *Command* interfészbe.
    - Minden *Command* objektumnak támogatnia kell a változtatásának visszavonását is az *Unexecute()* művelethez.
    - Minden *Command* objektum az *Execute()* során eltárolja tagváltozókban azt az állapotot, mely a visszavonáshoz szükséges.
  - *CommandProcessor* osztály bevezetése.
    - Eltárolja a már futtatott *Command* objektumokat, hogy ha később a parancs visszavonására kerül sor, rendelkezésre álljon a *Command* objektum.
    - A parancsok futtatását és visszavonását rajta keresztül végezzük (*ExecuteCommand()* és *UnexecuteLastCommand()* műveletek).

Összefoglalás

A *Command*-okból többet is végre akarsz hajtani és esetleg vissza is akarod őket vonni néha.

Megoldás: Végrehajtó osztály bevezetése és egy stack a *Command* objektumoknak.



Memento

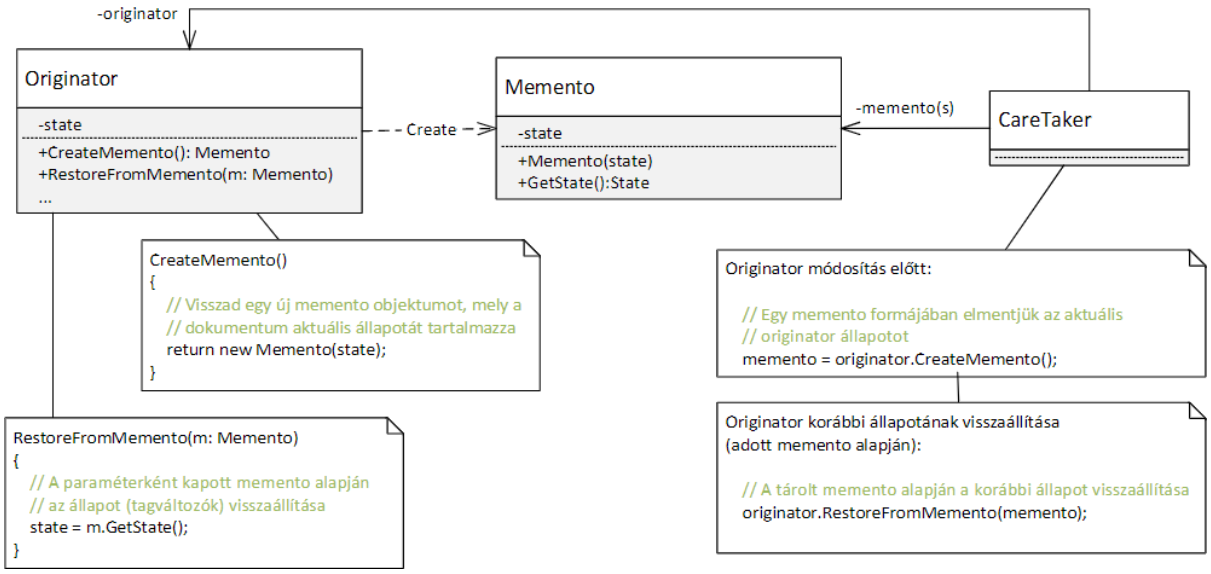
Jellemzők

- Célja az egységbezárás megsértése nélkül a külvilág számára elérhetővé tenni az objektum belső állapotát, vagyis pl. anélkül, hogy a védett változókat publikussá tennénk.
- Így az objektum állapota később visszaállítható.
- Bár a *Memento* a *Command*/*Command Processor* minták nélkül is használható, jellemzően ezekkel célszerű kombinálni.
- Lényege, hogy egy objektum (pl. dokumentum) adott állapotát egy *Memento* objektumba csomagoljuk be, és ilyen formában tesszük elérhetővé (a visszavonás megvalósításához).

Összefoglalás

El akarod menteni egy objektum állapotát, de úgy, hogy a belső, privát dolgai se vesszenek el. Kérsz tőle egy *memento*-t, amit te nem tudsz elolvasni, de el tudod neki küldeni később és az objektum az alapján vissza tudja állítani a saját állapotát.

Megoldás: *Memento*, ami képes leírni a belső állapotot és a függvények hozzá.



Adapter

Jellemzők

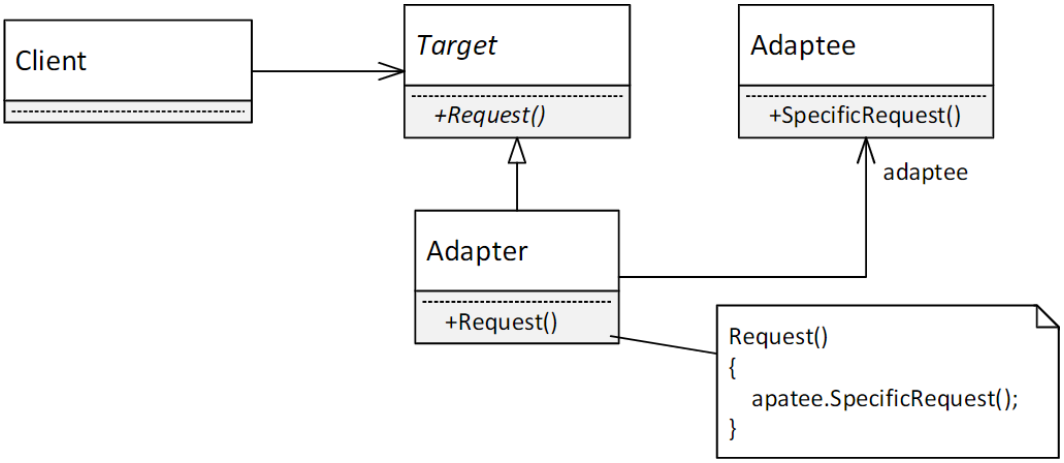
- Egy osztály interfészét olyan interfésszé konvertálja, amilyet a kliens vár.
- Lehetővé teszi olyan osztályok együttműködését, melyek egyébként az inkompatibilis interfészeik miatt nem tudnának együttműködni.
- Általánosságban két változata van: *Object Adapter* és *Class Adapter*.

Összefoglalás

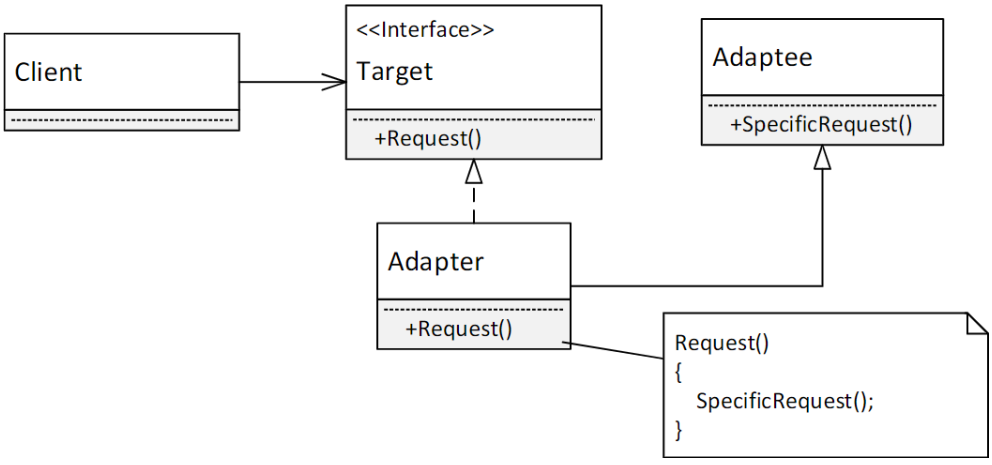
Van egy szolgáltatásod, amit egy külső komponens, vagy olyan kód akar elérni, ami nem kompatibilis a szolgáltatással, de ezt a külső komponenst/kódot nem tudod átírni valamiért.

Megoldás: Egy köztes osztály, ami konvertál a kellő formátumra.

Object Adapter: a becsomagolt objektum példányát tartalmazza (*delegált*).



Class Adapter: konkrét osztályt valósít meg (*öröklés*).



Composite

Jellemzők

- A rész-egész viszonyban álló objektumokat fastruktúrába rendezi.
- Egyik alapelve, hogy a kliensek számára lehetővé teszi az egyszerű (*elemi*) és összetett (*kompozit*) objektumok egységes kezelését.

Összefoglalás

Van egy viselkedésed, amit megvalósít egy osztály. Van viszont egy másik osztály is, ami szintén megvalósítja a viselkedést, de ráadásként még az első osztályt is tudja tartalmazni. Fa jellegű, hierarchikus tartalmazás jellemzően, ahol a lényeg, hogy az összetett és az elemi objektum egységesen kezelhető legyen, pl. egy könyvtár tud könyvtárat és fájlt is tartalmazni és egységes módon akarom őket törölni.

Megoldás: Egy közös interfész, egy-egy osztály, ami implementálja, valamint az összetett objektum még tartalmazza is az interfészt (az interfészt megvalósító objektumokat).