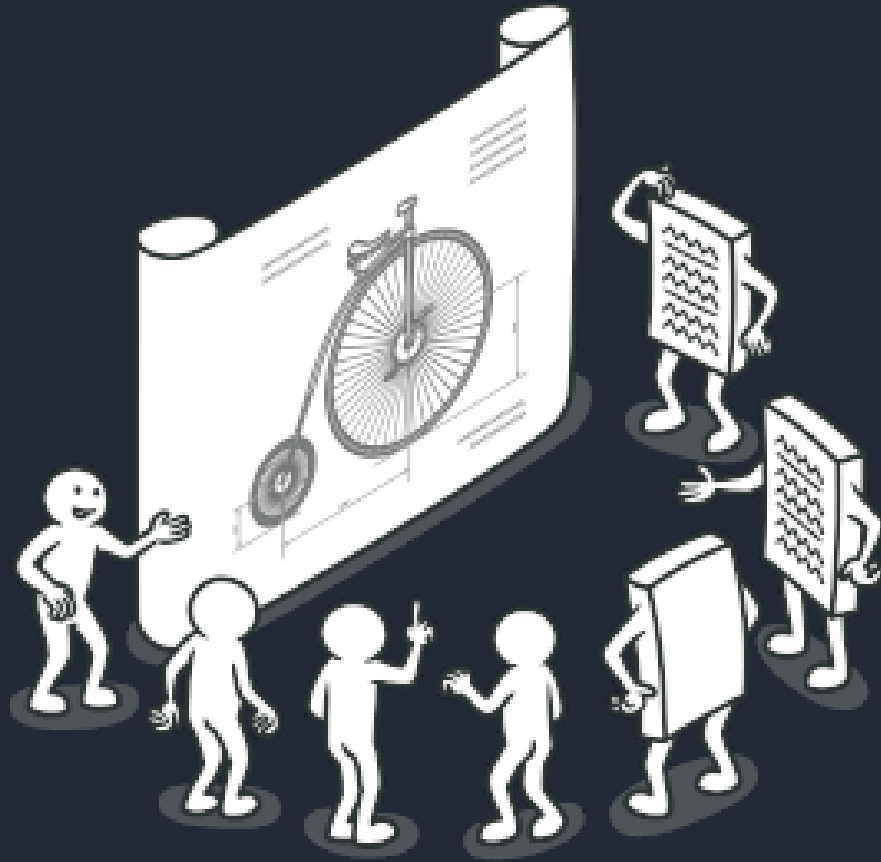


Tervezési minták egy OO programozási nyelvben

src: <https://refactoring.guru/design-patterns>

Tervezési minta

- A tervezési minták tipikus megoldások a szoftvertervezés során gyakran előforduló problémákra.
- Előre elkészített tervrajzokhoz hasonlítanak, amelyek segítségével megoldhatóak a visszatérő tervezési problémák.
- A tervezési minták követésével olyan megoldásokat alkalmazhatunk a kódban, ami az elvárásainknak megfelel.
- Az eredmény adott, az implementáció viszont nem.



- A tervezési minták gyakori problémákra kínálnak megoldást, a segítségükkel objektum orientált módon tudjuk előállítani a programunkat.
- A tervezési minták jól bevált és megbízható megközelítéseket nyújtanak, amelyeket nem lehet anélkül alkalmazni, hogy a saját kódunkra igazítanánk.
- A tervezési minták összetettségük, részletességük és a tervezett rendszer egészére való alkalmazhatóságuk mértéke alapján különböznek egymástól.

Creational kreatív minták ötleteket adnak az objektum létrehozási mechanizmusokra, amelyekkel flexibilis és újrafelhasználható kódot kapunk.

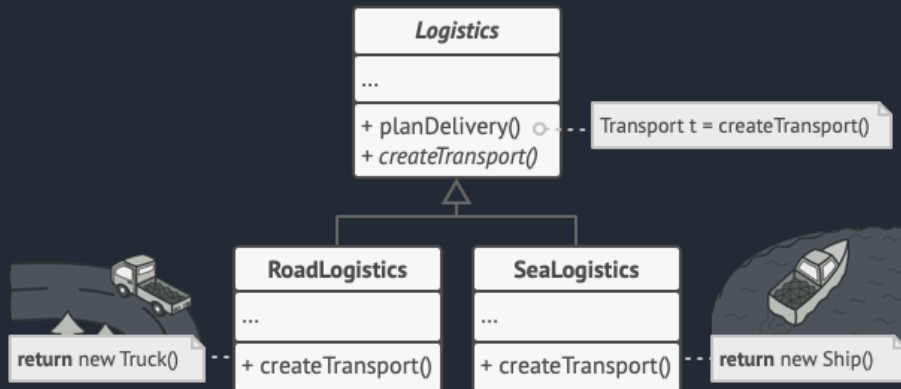
Structural szerkezeti minták segítségével az osztályainkat rendszerezni tudjuk

Behavioral viselkedésminták felosztják az objektumok közötti feladatokat az objektumok között



Creational minták

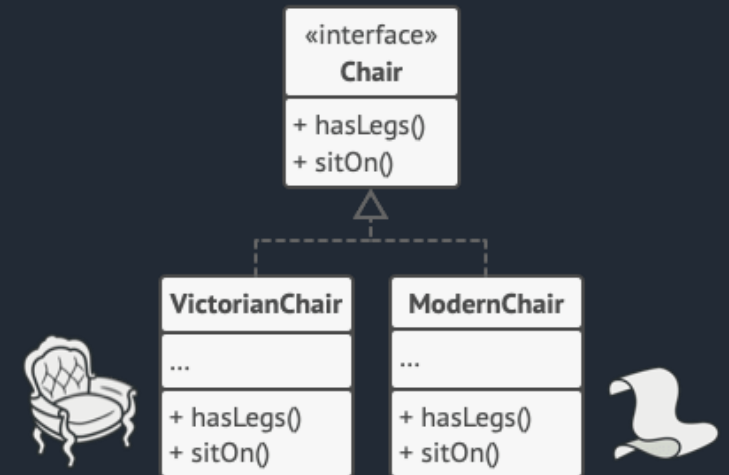
Factory



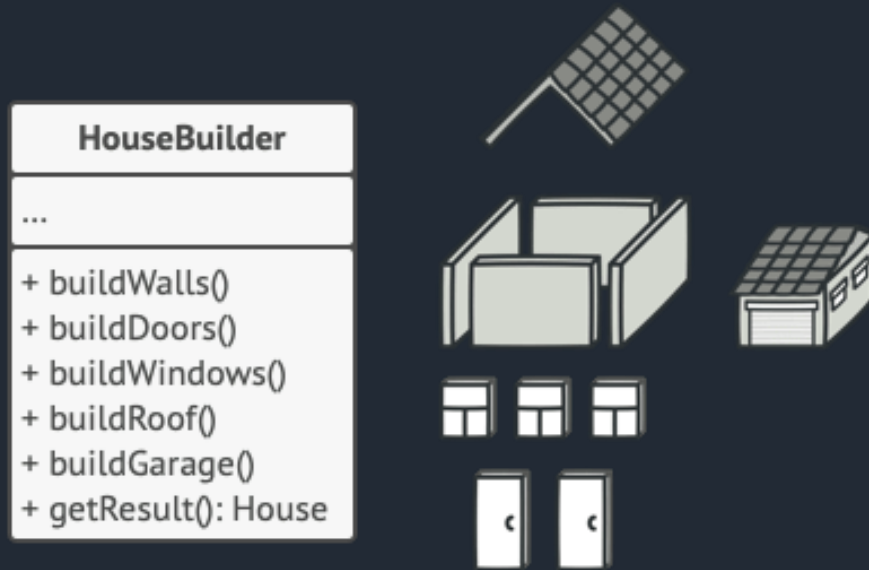
- A Factory egy olyan tervezési minta, amely interfészt biztosít egy felsőbb osztály objektumainak létrehozásához, de lehetővé teszi az alosztályok számára, hogy megváltoztassák a létrehozandó objektumok típusát. A felsőbb osztály hívja meg a többi osztály konstruktorait, a productokat
- A Factory módszer elválasztja a product készítő kódot az azt ténylegesen felhasználó kódtól. Ezért könnyebb a konstrukciós kódot a kód többi részétől függetlenül bővíteni.
- Egy új product típus hozzáadásához az alkalmazáshoz csak egy új alosztályt kell létrehozni, és felülírni benne a Factory metódusát.

Abstract Factory

- Az Abstract Factory egy olyan tervezési minta, amely lehetővé teszi kapcsolódó objektumcsaládok létrehozását anélkül, hogy konkrét osztályokat meg kellene adni.
- Ezzel a megközelítéssel az kliens kód nem függ a factoryk és a felhasználói felület elemeinek konkrét osztályaitól, amíg ezekkel az objektumokkal azok absztrakt interfészein keresztül dolgozik. Ez azt is lehetővé teszi, hogy az ügyfélkód más factorykat vagy UI-elemeket is támogasson, amelyeket a jövőben esetleg hozzáadhat.
- Ennek eredményeképpen nem kell minden alkalommal módosítani a kódot, amikor az UI elemeinek új variációját adjuk hozzá az alkalmazáshoz. Csak létre kell hozni egy új factoryt, amely ezeket az elemeket előállítja.



Builer



- A Builder minta azt javasolja, hogy az objektum konstrukciós kódját vegyük ki a saját osztályából, és helyezzük át külön objektumokba, úgynevezett builderekbe
- Az objektum létrehozásához használt lépések hívássorozatát egy külön osztályba, a directorba helyezhetjük. A director osztály határozza meg a lépések végrehajtásának sorrendjét, míg a builder biztosítja a lépések végrehajtását.
- E minta segítségével lépésről lépésre építhetünk objektumokat, és csak azokat a lépéseket használhatjuk, amelyekre valóban szükség van.

Prototype

- A Prototype egy olyan tervezési minta, amely lehetővé teszi a meglévő objektumok másolását anélkül, hogy a kód függővé válna azok osztályaitól.
- A minta a klónozási folyamatot a klónozendó objektumokhoz delegálja. A minta egy közös interfészt deklarál minden olyan objektum számára, amely támogatja a klónozást.
- Ez az interfész lehetővé teszi egy objektum klónozását anélkül, hogy a kódot az adott objektum osztályához kötnénk.
- Prototypeként előre elkészített, különböző módon konfigurált objektumokat használhatunk. Ahelyett, hogy egy olyan alosztályt példányosítanánk, amely megfelel valamilyen konfigurációnak, az ügyfél egyszerűen kereshet egy megfelelő prototípust, és klónozhatja azt.

Singleton

- A singleton egy olyan tervezési minta, amely lehetővé teszi, hogy egy osztálynak csak egy példánya legyen, miközben globális hozzáférési pontot biztosít ehhez a példányhoz.
- Ez a minta az osztály objektumainak létrehozására szolgáló minden más eszközt letilt, kivéve a speciális létrehozási módszert. Ez a metódus vagy létrehoz egy új objektumot, vagy visszaad egy már létezőt, ha az már létrejött.
- A Singleton minta garantálja, hogy egy osztályból csak egy példány létezik. Magán a Singleton osztályon kívül semmi sem helyettesítheti a példányt.

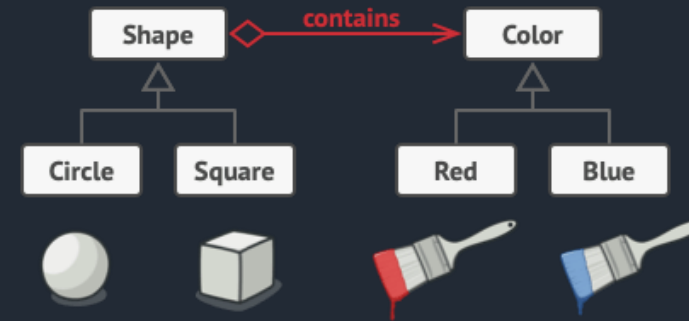


Structural minták

Adapter

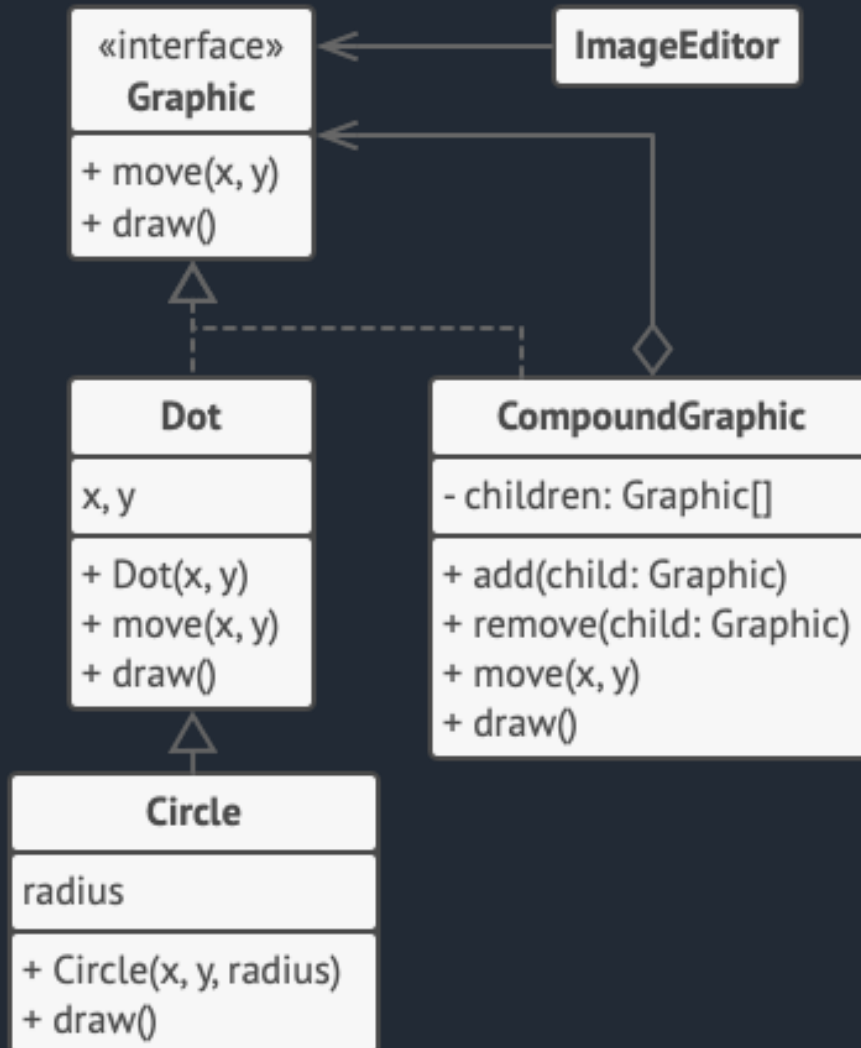
- Az Adapter egy olyan szerkezeti tervezési minta, amely lehetővé teszi az inkompatibilis interfészekkel rendelkező objektumok együttműködését.
- Az adapter az objektumot csomagolja be, hogy elrejtse a színfalak mögött zajló átalakítás összetettségét. A becsomagolt objektum nem is tud az adapterről.
- Az adapter lehetővé teszi egy olyan közbenső osztály létrehozását, amely fordítóként szolgál a kód és egy örökölt osztály, egy harmadik féltől származó osztály vagy bármely más interfésszel rendelkező osztály között.

Bridge



- A Bridge egy lehetővé teszi, hogy egy nagy osztályt vagy egy sor szorosan kapcsolódó osztályt két külön hierarchiára - absztrakció és implementáció - osszon fel, amelyek egymástól függetlenül fejleszthetők.
- Minél nagyobb egy osztály, annál nehezebb rájönni, hogyan működik, és annál hosszabb ideig tart a változtatás. A funkcionalitás egyik variációján végrehajtott változtatások az egész osztályban változtatásokat igényelhetnek, ami gyakran hibák elkövetését vagy néhány kritikus mellékhatások figyelmen kívül hagyását eredményezi.
- E minta segítségével a monolitikus osztályt több osztályhierarchiára oszthatjuk. Ezután az egyes hierarchiákban lévő osztályokat a többi hierarchia osztályaitól függetlenül módosíthatjuk. Ez a megközelítés leegyszerűsíti a kód karbantartását, és minimalizálja a meglévő kód megszakításának kockázatát.

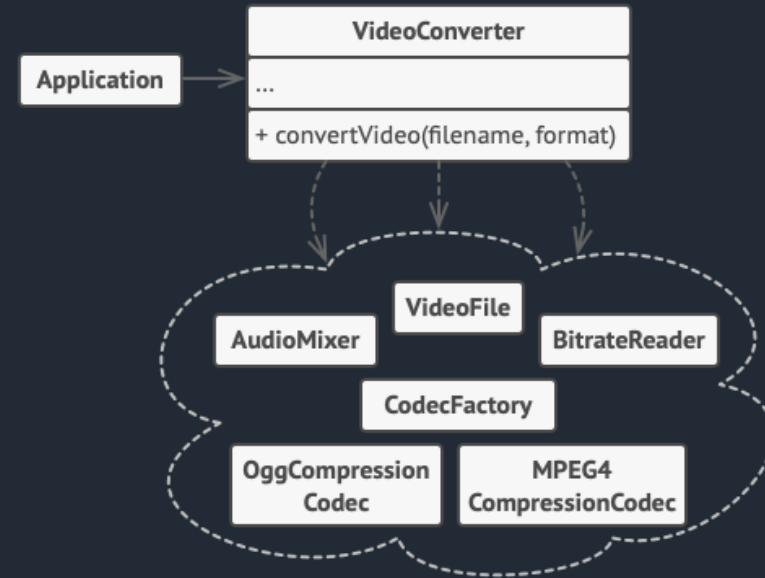
Composite



- A Composite lehetővé teszi, hogy objektumokat fa struktúrákba állítsuk össze, majd úgy dolgozzunk ezekkel a struktúrákkal, mintha önálló objektumok lennének.
- Az összes elem közös interfészen osztozik. Ezt az interfészt használva az ügyfélnek nem kell aggódnia azon objektumok konkrét osztálya miatt, amelyekkel dolgozik.

Decorator

- A Decorator lehetővé teszi új viselkedések objektumokhoz való csatolását azáltal, hogy ezeket az objektumokat a viselkedéseket tartalmazó speciális wrapper objektumokba helyezi.
- Segítségével az üzleti logikát rétegekbe strukturálhatjuk, minden réteghez létrehozhatunk egy decorator-t, és futásidőben összeállíthatjuk az objektumokat e logika különböző kombinációival. Az ügyfélkód ugyanúgy kezelheti ezeket az objektumokat, mivel mind egy közös interfészt követnek.



Facade


- A Facade egy szerkezeti tervezési minta, amely egyszerűsített felületet biztosít egy könyvtár, egy keretrendszer vagy bármely más összetett osztálykészlet számára.
- Korlátozott funkcionalitást biztosíthat a közvetlen alrendszerrel való munkához képest, azonban csak azokat a funkciókat tartalmazza, amelyek az ügyfeleket valóban érdeklik.
- Egy alrendszer rugalmasabbá válhat és könnyebben újrafelhasználható különböző kontextusokban, de az ügyféltől megkövetelt konfigurációs és alapkód mennyisége egyre nagyobb lesz. A Facade úgy próbálja orvosolni ezt a problémát, hogy rövidített utat biztosít az alrendszer leggyakrabban használt, a legtöbb ügyfél igényeinek megfelelő funkcióihoz.

Flyweight

- A Flyweight lehetővé teszi, hogy több objektum férjen el a rendelkezésre álló RAM memóriában azáltal, hogy az állapot közös részeit több objektum között osztja meg ahelyett, hogy az összes adatot az egyes objektumokban tartaná.
- E minta azt javasolja, hogy az objektumon belül ne tároljuk tovább a külső állapotot. Ehelyett át kell adnunk ezt az állapotot az ettől függő specifikus metódusoknak.

Proxy

- A Proxy egy strukturális tervezési minta, amely lehetővé teszi, hogy egy másik objektum helyettesítését vagy helytartóját biztosítsuk.
- A proxy ellenőrzi az eredeti objektumhoz való hozzáférést, lehetővé téve, hogy valamit elvégezzen, mielőtt vagy miután a kérés eljut az eredeti objektumhoz.
- Mivel a Proxy ugyanazt az interfészt valósítja meg, mint az eredeti osztály, átadható bármely olyan ügyfélnek, amely valódi service objektumot vár.
- Ahelyett, hogy az objektumot az alkalmazás indításakor hoznánk létre, késleltethetjük az objektum inicializálását egy olyan időpontra, amikor valóban szükség van rá.



Behavioral minták

Chain of Responsibility

- A lánc egy viselkedési tervezési minta, amely lehetővé teszi, hogy a kéréseket kezelők láncolatán keresztül továbbítsa.
- A kérés fogadásakor minden egyes kezelő eldönti, hogy feldolgozza-e a kérést, vagy átadja a lánc következő kezelőjének.
- A minta lehetővé teszi, hogy több kezelőt egy láncba kapcsoljunk, és egy kérés fogadásakor minden egyes kezelőtől "megkérdezzük", hogy képes-e feldolgozni azt. Így minden kezelő kap esélyt a kérés feldolgozására.

Command

- A Command egy viselkedési tervezési minta, amely a kérést egy önálló objektummá alakítja, amely a kéréssel kapcsolatos összes információt tartalmazza.
- Ez az átalakítás lehetővé teszi a kérések átadását metódusargumentumként, a kérés végrehajtásának késleltetését vagy sorba állítását, valamint a visszavonható műveletek támogatását.
- A minta egy adott metódushívást önálló objektummá alakíthat. Ez a változás sok érdekes felhasználási lehetőséget nyit meg: parancsokat adhatunk át metódusargumentumként, tárolhatjuk őket más objektumokon belül, futásidőben kapcsolhatjuk a kapcsolt parancsokat, stb.

Iterator

- Az Iterator egy viselkedéses tervezési minta, amely lehetővé teszi egy gyűjtemény elemeinek átjárását anélkül, hogy felfedné a mögöttes reprezentációt (lista, verem, fa stb.).
- Az iterátor egy összetett adatstruktúrával való munka részleteit zárja magába, és az ügyfél számára számos egyszerű módszert biztosít a gyűjtemény elemeinek eléréséhez.
- Bár ez a megközelítés nagyon kényelmes az ügyfél számára, ugyanakkor megvédi a gyűjteményt az óvatlan vagy rosszindulatú műveletektől, amelyeket az ügyfél akkor tudna végrehajtani, ha közvetlenül a gyűjteménnyel dolgozna.

Mediator

- A Mediator egy viselkedési tervezési minta, amellyel csökkenthetjük az objektumok közötti függőségeket.
- A minta korlátozza az objektumok közötti közvetlen kommunikációt, és arra kényszeríti őket, hogy csak egy közvetítő objektumon keresztül működjenek együtt.
- A minta lehetővé teszi, hogy az osztályok közötti összes kapcsolatot egy külön osztályba vonjuk ki, elszigetelve egy adott komponens bármilyen változását a többi komponenstől.

Memento, Snapshot

- A Memento lehetővé teszi egy objektum korábbi állapotának mentését és visszaállítását anélkül, hogy felfedné a végrehajtás részleteit.
- A minta lehetővé teszi, hogy az objektum állapotáról teljes másolatokat készítsünk, beleértve a privát mezőket is, és ezeket az objektumtól elkülönítve tároljuk.
- Bár a legtöbbször a "visszavonás" felhasználási esetnek köszönhetően emlékeznek erre a mintára, tranzakciók esetén is nélkülözhetetlen



Observer

- Az Observer egy minta, amely lehetővé teszi egy olyan feliratkozási mechanizmus definiálását, amellyel több objektumot értesíthetünk minden olyan eseményről, amely a megfigyelt objektummal történik.
- Az Observer mintát akkor használjuk, ha egy objektum állapotának megváltoztatása más objektumok megváltoztatását is szükségessé teheti, és az objektumok tényleges halmaza előre nem ismert, vagy dinamikusan változik.

State

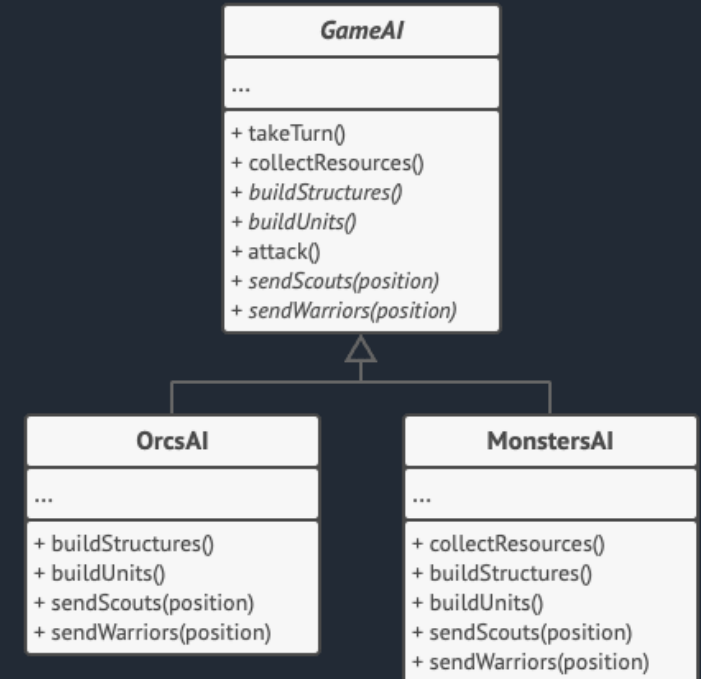
- A State lehetővé teszi, hogy egy objektum megváltoztassa a viselkedését, amikor a belső állapota megváltozik. Úgy tűnik, mintha az objektum megváltoztatta volna az osztályát.
- Ezt a mintát akkor használjuk, ha olyan objektumunk van, amely az aktuális állapotától függően másképp viselkedik, az állapotok száma sok, és az állapot-specifikus kód gyakran változik.
- A minta azt javasolja, hogy az összes állapot-specifikus kódot különálló osztályok halmazába vonjuk. Ennek eredményeként egymástól függetlenül tudunk új állapotokat hozzáadni vagy a meglévőket módosítani, csökkentve ezzel a karbantartási költségeket.

Strategy

- A Strategy egy viselkedési tervezési minta, amely lehetővé teszi, hogy algoritmusok egy családját definiáljuk, mindegyiküket külön osztályba helyezzük, és objektumaikat felcserélhetővé tegyük.
- A minta lehetővé teszi, hogy közvetve megváltoztassuk az objektum viselkedését futás közben azáltal, hogy különböző alobjektumokkal társítjuk, amelyek különböző módon végezhetnek el bizonyos részfeladatokat.
- A stratégia minta lehetővé teszi, hogy a változó viselkedést egy külön osztályhierarchiába vonjuk ki, és az eredeti osztályokat egyesítsük egybe, ezáltal csökkentve a duplikált kódot.

Template

- A Template módszer egy viselkedési tervezési minta, amely meghatározza egy algoritmus vázát a szuperosztályban, de lehetővé teszi az alosztályok számára, hogy felülírják az algoritmus bizonyos lépéseit anélkül, hogy megváltoztatnák a struktúráját.
- A minta lehetővé teszi, hogy egy monolitikus algoritmust egyedi lépések sorozatává alakítsunk, amelyek könnyen bővíthetők alosztályokkal, miközben a szuperosztályban definiált struktúra érintetlen marad.
- A mintát akkor használjuk, ha több olyan osztályunk is van, amelyek csaknem azonos algoritmusokat tartalmaznak, kisebb eltérésekkel. Előfordulhat, hogy az algoritmus változása esetén az összes osztályt módosítani kell.



Visitor

- A Visitor lehetővé teszi az algoritmusok elkülönítését azoktól az objektumoktól, amelyeken működnek.
- A minta lehetővé teszi, hogy egy műveletet különböző osztályú objektumok halmazán hajtsunk végre azáltal, hogy egy visitor objektum ugyanazon művelet több változatát is végrehajtja, amelyek megfelelnek az összes célosztálynak.
- A minta lehetővé teszi, hogy az alkalmazás elsődleges osztályai jobban összpontosítsanak a fő feladataikra azáltal, hogy az összes többi viselkedést egy sor visitor osztályba vonjuk.