Návrhové vzory

Návrhové vzory jsou nedílnou součástí znalostní výbavy každého profesionála v oblasti tvorby softwaru. Umění aplikovat návrhové vzory je dnes stejně důležité jako znát knihovny a syntaxi příslušného programovacího jazyka. Návrhové vzory popisují mnohokrát vyzkoušená jednoduchá a elegantní řešení různých dílčích problémů při tvorbě softwaru. O návrhových vzorech vychází řada knih.

Každý návrhový vzor má následující strukturu:

- **Název vzoru**. Název je krátké nejlépe jednoslovní označení.
- **Popis problému**, který návrhový vzor řeší.
- **Řešení**, které návrhový vzor přináší.
- **Důsledky použití** vzoru.

Většina publikací návrhové vzory člení do následujících kategorií:

- **Tvořivé vzory**.
- **Strukturální vzory**
- **Behaviorální vzory**

Tvořivé (creational) vzory

- Abstract Factory
- Builder
- Factory Method
- Prototype
- Singleton

Tvořivé vzory jsou vyzkoušené postupy na řešení problémů s vytvářením objektů.

Strukturální (structural) vzory

- Adapter
- Bridge
- Composite
- Decorator
- Facade
- Flyweight
- Proxy

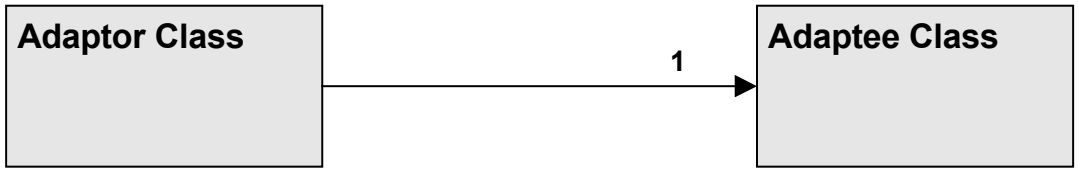
Strukturální vzory jsou vyzkoušené postupy na řešení problémů při zpracování dat.

Behaviorální (behavioral) vzory

- Chain of Responsibility
- Command
- Interpreter
- Iterator
- Mediator
- Memento
- Observer
- State
- Strategy
- Template Method
- Visitor

Behaviorální vzory jsou vyzkoušené postupy na řešení problémů s činností objektů a se změnami objektů v čase.

Adaptér (Adaptor)



Adaptér (anglicky Adaptor nebo jiným názvem Wrapper) je strukturální vzor. Tento vzor převádí protokol nějakého objektu na jiný protokol. Tím umožňuje využití objektů, které by jinak v datovém modelu nemohly pracovat.

řešený problém

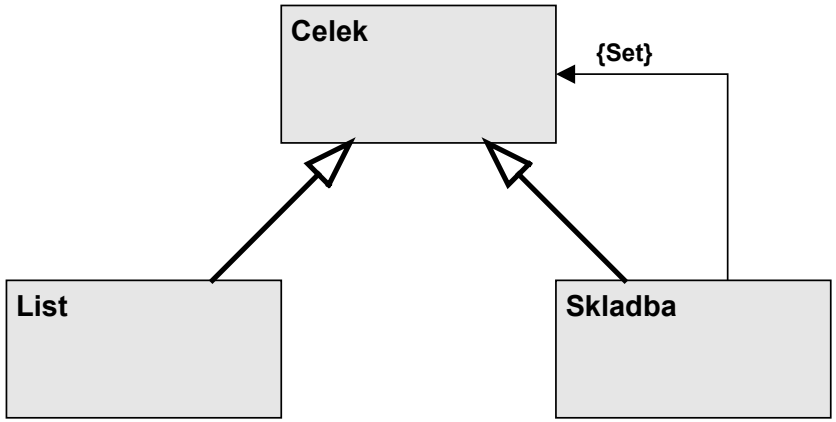
Někdy se stane, že bychom rádi použili nějaký objekt, o kterém víme, že má vlastnosti, které potřebujeme, ale náš systém od toho objektu požaduje jiný protokol, než ten, co už daný objekt má.

rozbor

Adaptér se používá, když potřebujeme použít již hotový, ale poněkud odlišný objekt nebo když potřebujeme nějaký objekt využít v různých situacích různými způsoby a proto ho musíme „obalit" různými adaptéry.

Tento vzor je podobný dekorátoru, ale na rozdíl od něj může dát objektu úplně jiný protokol. Dekorátor totiž protokol jen doplňuje o nové atributy.

Skladba (Composite)



Skladba (anglicky Composite) je strukturální vzor. Tento vzor je návodem k implementaci hierarchických datových struktur.

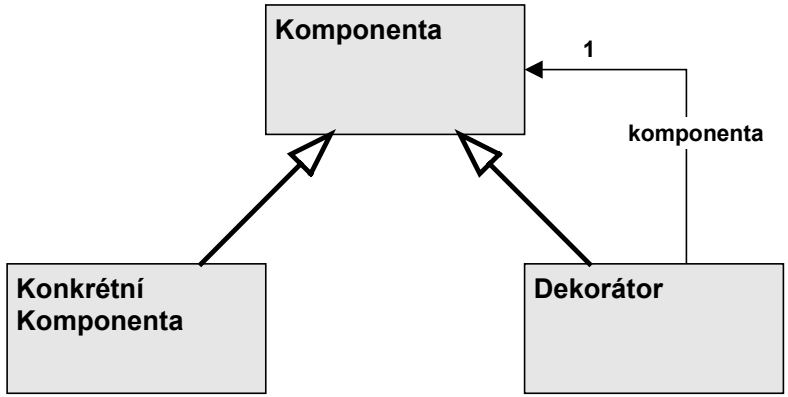
řešený problém

Někdy se stane, že pracujeme s objekty, které se skládají z dalších objektů a ty opět z dalších objektů a tak dále. Zároveň nemůžeme předem odhadnout hloubku takového hierarchického skládání. Tento problém řeší vzor Skladba. Třída Celek je abstraktní třída bez instancí, ale dědí z ní třídy, které mají v datovém modelu instance. List je element, který dále nic neobsahuje, Skladba je element, který obsahuje další elementy (listy i jiné skladby).

rozbor

Tento vzor se často používá v kombinaci s dekorátorem. Pomocí dekorátoru se totiž výhodně implementují specifické vlastnosti listů ve skladbě.

Dekorátor (Decorator)



Dekorátor (anglicky Decorator) je strukturální vzor. Pomocí dekorátoru můžeme přidávat objektům vlastnosti aniž bychom rušili jejich stávající vlastnosti.

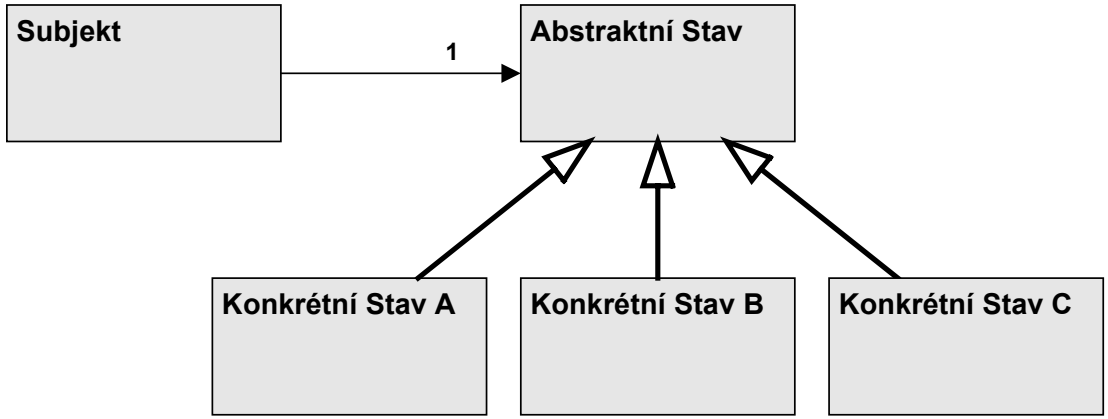
řešený problém

Někdy potřebujeme přidat vlastnosti jen některým objektům dané třídy přičemž ostatní instance dané třídy zůstávají beze změny. Dekorátorem je podtřída, jejíž instance obsahují jako složku konkrétní komponentu. Toto skládání je podobné adaptéru, ale vlivem dědění přebíráme také protokol nadtřídy.

rozbor

Dekorátor je svojí funkcí velmi podobný adaptéru, ale nepřekrývá původní protokol ošetřovaného objektu. Na rozdíl od prostého dědění má tu vlastnost, že toto doplnění funkčnosti můžeme uskutečnit jen u některých instancí původní třídy. Kdybychom použili dědění, tak bychom vytvořením nové třídy vyrobili nové chování pro všechny instance této třídy a tak i pro ty objekty, které to nepotřebují.

Stav (State)



Stav (anglicky State Pattern) je behaviorální vzor. Umožňuje za chodu systému měnit vlastnosti objektu tak, že se z pohledu uživatele zdá, že objekt mění svoji třídu.

řešený problém

Často se stává, že nějaké objekty v datovém modelu zobrazují proměnlivé entity reálného světa. Bez použití tohoto vzoru bychom při každé takové změně museli původní objekt smazat a namísto něj vytvořit nový jiný objekt. Vzor Stav ale tyto proměnu zapouzdřuje do vloženého objektu. Atributy, které změně nepodléhají, zůstávají v původním objektu, ale atributy, které jsou proměnlivé, jsou uloženy ve „vnitřním" objektu. Navenek se celá struktura chová jako jediný objekt.

rozbor

Stav je podobný adaptéru v tom, že i zde je dvojice objektů (vnější a vnitřní), která se navenek jeví jako jediný objekt. Rozdíl je ale v tom, že u adaptéru je hlavním nosičem dat vložený objekt, ale u stavu je vložený objekt v roli „přídavných" atributů, které se mohou měnit v čase.

… a ještě jeden „návrhový vzor"

DIVIDE ET IMPERA

rozděl a panuj

Rozděl a panuj není návrhový vzor podle klasické definice, protože jde o postup, který byl používán již před OOP.

řešený problém

Máme-li několik variant operací nad jednou třídou objektů, tak by se tato třída měla rozdělit na více vzájemně polymorfních tříd tak, aby každá z nich vlastnila jen jednu variantu operace.

oblast použití

- tvorba nových datových typů
- double dispatching

Double Dispatching

řešený problém

Je třeba řešit situaci, kdy máme nějakou operaci s **N** různými objekty, které mohou být v **M** různých situacích, což vede k **N.M** variantám této operace. Double dispatching je návod, jak využít sebeidentifikace objektů k tomu, abychom nemuseli použít větvení ani jiné strukturované konstrukce při formulaci algoritmu diskutované operace.

rozbor

Místo jedné supermetody budeme potřebovat **N** + **N.M** metod. Celková velikost kódu bude ale stejná. Navíc získáme výhodu rozšiřitelnosti ve směru **N** i **M** bez potřeby měnit stávající kód.

DIVIDE ET IMPERA