NÁVRHOVÉ VZORY

Přednáška 3

CREATIONAL DESIGN PATTERNS

- Creational Design Patterns poskytují různé mechanismy pro vytváření objektů, které zvyšují flexibilitu a umožňují znovupoužití existujícího kódu

BUILDER

- Builder je tvořivý- creational návrhový vzor, který umožňuje vytvářet složité objekty krok za krokem.
- Tento vzor dovoluje vyrábět různé typy a reprezentace objektu pomocí stejného konstrukčního kódu.

CO ŘEŠÍ?

- Představte si složitý objekt, který vyžaduje zdlouhavou, krok za krokem prováděnou inicializaci mnoha polí a vnořených objektů.
- Takový inicializační kód bývá obvykle ukrytý uvnitř **obrovského konstruktoru** s mnoha parametry — nebo ještě hůře, roztříštěný napříč klientským kódem.
- Například si představme, jak bychom mohli vytvořit objekt House (dům).
  - K postavení jednoduchého domu potřebujete vytvořit čtyři stěny a podlahu, nainstalovat dveře, zasadit pár oken a postavit střechu.
  - Ale co když chcete větší, světlejší dům se zahradou a dalšími vymoženostmi (například topením, vodovodem a elektrickým rozvodem)?
  - Nejjednodušším řešením by bylo **rozšířit základní třídu House** a vytvořit **sadu podtříd**, které by pokrývaly všechny možné kombinace parametrů.
  - Nakonec byste však skončili s **velkým množstvím potomků** a každý nový parametr (například typ verandy) by vyžadoval, aby se tato hierarchie dále rozrůstala.
- Existuje ale i jiný přístup, který nezahrnuje množení potomků.
- Můžete vytvořit obrovský konstruktor přímo v základní třídě House, který bude mít všechny možné parametry určující podobu domu.
- Tento přístup sice odstraní potřebu podtříd, ale přináší jiný problém — ve **většině případů většina parametrů nebude využita**, což způsobí, že volání konstruktoru bude velmi těžkopádné.
- Například jen malé procento domů má bazén, takže parametry týkající se bazénu budou v devíti z deseti případů zbytečné.

ŘEŠENÍ

- Návrhový vzor Builder (Stavitel) navrhuje, abyste **oddělili kód pro vytváření objektu z jeho vlastní třídy** a přesunuli ho do samostatných objektů nazývaných stavitelé (builders).
- Tento vzor **organizuje** konstrukci objektu do **sady kroků** (např. buildWalls, buildDoor atd.).
- Pro vytvoření objektu pak spustíte sérii těchto kroků na objektu stavitele.
- Důležité je, že **nemusíte volat všechny kroky** — můžete volat pouze ty, které jsou potřeba pro vytvoření konkrétní konfigurace objektu.
- Některé kroky konstrukce mohou vyžadovat různou implementaci, pokud je potřeba vytvořit různé podoby výsledného produktu.
- Například stěny chaty mohou být ze dřeva, zatímco stěny hradu musí být z kamene.
- V takovém případě můžete vytvořit několik různých tříd stavitelů, které implementují stejnou sadu stavebních kroků, ale každý svým vlastním způsobem.
- Tyto stavitele pak můžete použít v konstrukčním procesu (tedy ve stanoveném pořadí volání stavebních kroků) k vytvoření různých typů objektů.
- Například si představte jednoho stavitele, který staví vše ze dřeva a skla, druhého, který staví z kamene a železa, a třetího, který používá zlato a diamanty.
- Voláním stejné sady kroků získáte z prvního běžný dům, z druhého malý hrad a z třetího palác.
- Tento přístup však funguje pouze tehdy, pokud klientský kód, který volá jednotlivé stavební kroky, umí komunikovat se staviteli prostřednictvím společného rozhraní.

DIRECTOR

- Můžete zajít ještě dál a vytáhnout sekvenci volání stavebních kroků, které používáte k vytvoření produktu, do samostatné třídy nazývané ředitel (Director).
- Třída Director určuje pořadí, v jakém se mají stavební kroky provádět, zatímco stavitel (Builder) poskytuje jejich konkrétní implementaci.
- Mít třídu Director v programu není povinné – stavební kroky můžete vždy volat ve správném pořadí přímo z klientského kódu.
- Nicméně třída Director je vhodné místo pro uložení různých stavebních postupů, které pak můžete znovu použít napříč programem.
- Kromě toho třída Director zcela **skryje detaily konstrukce** produktu před klientským kódem.
- Klient pouze přiřadí stavitele řediteli, spustí konstrukci pomocí ředitele a poté získá výsledek od stavitele.

VYUŽITÍ

- Použijte návrhový vzor Builder (Stavitel) k odstranění tzv. „teleskopického konstruktoru“.
- Řekněme, že máte konstruktor s deseti volitelnými parametry.
- Volání takového „monstra“ je velmi nepraktické, proto konstruktor přetížíte a vytvoříte několik kratších verzí s menším počtem parametrů.
- Tyto konstruktory stále odkazují na hlavní konstruktor a předávají výchozí hodnoty pro všechny vynechané parametry.

```
class Pizza {
    Pizza(int size) { ... }
    Pizza(int size, boolean cheese) { ... }
    Pizza(int size, boolean cheese, boolean pepperoni) { ... }
    // ...
}
```

- Vytvoření takového „monstra“ je možné pouze v jazycích, které podporují přetěžování metod, jako jsou C# nebo Java.
- Návrhový vzor Builder umožňuje stavět objekty krok za krokem, přičemž použijete pouze ty kroky, které opravdu potřebujete. Po implementaci tohoto vzoru už nemusíte cpát desítky parametrů do konstruktorů.

VYUŽITÍ 2

- Použijte Builder, když chcete, aby váš kód dokázal vytvářet různé podoby nějakého produktu (například kamenné a dřevěné domy).
- Vzor Builder se hodí, když konstrukce různých podob produktu zahrnuje podobné kroky, lišící se jen detaily.
- Základní rozhraní stavitele (base builder interface) definuje všechny možné konstrukční kroky a konkrétní stavitelé (concrete builders) tyto kroky implementují pro vytvoření konkrétních podob produktu.
- Mezitím třída Director určuje pořadí konstrukce.

VYUŽITÍ 3

- Použijte Builder k vytvoření Composite stromů nebo jiných složitých objektů.
- Vzor Builder umožňuje stavět produkty krok za krokem.
- Některé kroky lze odložit bez toho, aby se poškodil výsledný produkt.
- Dokonce můžete kroky volat i rekurzivně, což se hodí, pokud potřebujete vytvořit strom objektů.
- Stavitel neodhaluje nedokončený produkt během provádění kroků konstrukce.
- Tím se zabraňuje tomu, aby klientský kód získal neúplný výsledek.

JAK IMPLEMENTOVAT?

1. Ujistěte se, že dokážete jasně definovat společné konstrukční kroky pro vytváření všech dostupných podob produktu. Jinak nebude možné pokračovat v implementaci vzoru.
2. Tyto kroky deklarujte v základním rozhraní stavitele (base builder interface).
3. Vytvořte konkrétní třídu stavitele pro každou podobu produktu a implementujte v ní jejich konstrukční kroky.
    1. Nezapomeňte implementovat metodu pro získání výsledku konstrukce.
    2. Důvod, proč tuto metodu nelze deklarovat přímo v rozhraní stavitele, je ten, že různí stavitelé mohou vytvářet produkty, které nemají společné rozhraní.
    3. Proto nevíte, jaký by měl být návratový typ takové metody.
    4. Pokud ale pracujete s produkty ze společné hierarchie, lze metodu pro získání výsledku bezpečně přidat do základního rozhraní.
4. Zvažte vytvoření třídy Director. Ta může zapouzdřit různé způsoby konstrukce produktu při použití stejného objektu stavitele.
5. Klientský kód vytváří objekty stavitele i ředitele.
    1. Před zahájením konstrukce musí klient předat objekt stavitele řediteli.
    2. Obvykle klient toto provede pouze jednou, prostřednictvím parametrů konstruktoru třídy ředitele.
    3. Ředitel pak používá stavitele ve všech dalších krocích konstrukce.
    4. Existuje také alternativní přístup, kdy je stavitel předán konkrétní metodě pro konstrukci produktu v řediteli.
6. Výsledek konstrukce lze získat přímo od ředitele pouze pokud všechny produkty následují stejné rozhraní.V opačném případě by měl klient výsledek získat přímo od stavitele.

PRO A PROTI

Výhody

- Můžete stavět objekty krok za krokem, odkládat některé kroky konstrukce nebo volat kroky rekurzivně.
- Stejný konstrukční kód lze znovu použít při vytváření různých podob produktu.
- Princip jediné odpovědnosti (Single Responsibility Principle): můžete izolovat složitý konstrukční kód od obchodní logiky produktu.

Nevýhody

- Celková složitost kódu však narůstá, protože vzor vyžaduje vytvoření několika nových tříd.

VZTAHY S JINÝMI VZORY

- Mnoho návrhů začíná použitím Factory Method (méně složité a snadněji přizpůsobitelné pomocí podtříd) a postupně se vyvíjí směrem k Abstract Factory, Prototype nebo Builder (flexibilnější, ale složitější).
- Builder se zaměřuje na konstrukci složitých objektů krok za krokem.Abstract Factory se specializuje na vytváření rodin souvisejících objektů.Abstract Factory vrací produkt okamžitě, zatímco Builder vám umožňuje spustit další konstrukční kroky před získáním produktu.
- Builder můžete použít při vytváření složitých Composite stromů, protože můžete naprogramovat jeho konstrukční kroky tak, aby fungovaly rekurzivně.
- Builder lze kombinovat s Bridge: třída Director hraje roli abstrakce, zatímco různí stavitelé (builders) působí jako implementace.
- Abstract Factory, Builder i Prototype lze implementovat jako Singletony.

PŘÍKLAD

- Postupná výroba automobilůV tomto příkladu návrhový vzor Builder umožňuje krok za krokem stavět různé modely automobilů.
- Příklad také ukazuje, jak Builder produkuje produkty různých druhů (např. návod k obsluze auta) pomocí stejných stavebních kroků.
- Ředitel (Director) řídí pořadí konstrukce.
- Ví, které kroky volat, aby vznikl konkrétní model auta.
- Pracuje se staviteli pouze prostřednictvím jejich společného rozhraní, což umožňuje předávat řediteli různé typy stavitelů.
- Konečný výsledek se získává z objektu stavitele, protože ředitel nemůže vědět, jaký typ produktu vznikne.
- Pouze objekt Builder přesně ví, co staví.

PROTOTYPE

- Prototype je tvořivý (creační) návrhový vzor, který vám umožňuje kopírovat existující objekty, aniž by byl váš kód závislý na jejich třídách.

CO ŘEŠÍ?

- Řekněme, že máte nějaký objekt a chcete vytvořit jeho přesnou kopii.
- Jak byste to udělali?
- Nejprve musíte vytvořit nový objekt stejné třídy.
- Poté musíte projít všechna pole původního objektu a zkopírovat jejich hodnoty do nového objektu.
- Ale je tu jeden háček.
- Ne všechny objekty lze tímto způsobem zkopírovat, protože některá pole objektu mohou být privátní a nepřístupná zvenčí.
- A to není vše — přímý přístup má ještě další problém.
- Protože musíte znát třídu objektu, abyste mohli vytvořit jeho kopii, váš kód se stává závislým na této třídě.
- další komplikace: Někdy znáte pouze rozhraní, které objekt implementuje, ale ne jeho konkrétní třídu — například v situaci, kdy parametr metody přijímá libovolné objekty, které dodržují určité rozhraní.

ŘEŠENÍ

- Vzor Prototype (Prototyp) přenáší proces klonování na samotné objekty - delegace
- Tento vzor definuje společné rozhraní pro všechny objekty, které podporují klonování.
- Díky tomuto rozhraní můžete klonovat objekt, aniž by byl váš kód závislý na jeho konkrétní třídě.
- Obvykle toto rozhraní obsahuje pouze jednu metodu – clone().
- Implementace metody clone je ve všech třídách velmi podobná.
- Metoda vytvoří nový objekt aktuální třídy a zkopíruje všechny hodnoty polí ze starého objektu do nového.
- Můžete dokonce kopírovat i privátní pole, protože většina programovacích jazyků umožňuje objektům přistupovat k privátním polím jiných objektů stejné třídy.
- Objekt, který podporuje klonování, se nazývá prototyp.
- Pokud mají vaše objekty desítky polí a stovky možných konfigurací, může být klonování vhodnou alternativou k dědičnosti.
- Jak to funguje:Vytvoříte sadu objektů, nakonfigurovaných různými způsoby.
- Kdykoli potřebujete nový objekt, který se některému z nich podobá, jednoduše zkopírujete prototyp místo toho, abyste vytvářeli nový objekt od začátku.

POUŽITÍ VZORU

- Pokud váš kód nemá záviset na konkrétních třídách objektů, které potřebujete kopírovat.
  - To nastává často, když váš kód pracuje s objekty, které jsou vám předávány z externího (3rd-party) kódu prostřednictvím nějakého rozhraní.
  - Konkrétní třídy těchto objektů nejsou známé a ani na nich nemůžete být závislí, i kdybyste chtěli.
  - Vzor Prototype poskytuje klientskému kódu obecné rozhraní pro práci se všemi objekty, které podporují klonování.
  - Toto rozhraní dělá klientský kód nezávislým na konkrétních třídách objektů, které klonuje.
- pokud chcete snížit počet podtříd, které se liší pouze způsobem, jakým inicializují své objekty.
  - Představte si složitou třídu, která vyžaduje náročnou konfiguraci před použitím.
  - Existuje několik běžných způsobů, jak tuto třídu nakonfigurovat, a tento kód je roztříštěn po celé aplikaci.
  - Abyste odstranili duplicitu, vytvoříte několik podtříd, do jejichž konstruktorů umístíte potřebný inicializační kód.
  - Tím sice problém s duplicitou vyřešíte, ale zároveň získáte spoustu zbytečných podtříd.
  - Vzor Prototype vám umožní použít sadu předem vytvořených objektů, nakonfigurovaných různými způsoby, jako prototypy.
  - Místo vytváření nové instance podtřídy odpovídající konkrétní konfiguraci může klient jednoduše vyhledat vhodný prototyp a zkopírovat ho.

JAK IMPLEMENTOVAT?

1. Vytvořte rozhraní prototypu a deklarujte v něm metodu clone().Nebo prostě přidejte tuto metodu do všech tříd existující hierarchie, pokud nějakou máte.
2. Třída prototypu musí definovat alternativní konstruktor, který přijímá objekt téže třídy jako argument.Konstruktor musí zkopírovat hodnoty všech polí definovaných v třídě z předaného objektu do nově vytvořené instance.Pokud měníte podtřídu, musíte volat konstruktor nadřazené třídy, aby nadtřída mohla zpracovat klonování svých privátních polí.
3. Pokud váš programovací jazyk nepodporuje přetěžování metod, nebudete schopni vytvořit samostatný „prototypový" konstruktor.Kopírování dat objektu do nově vytvořeného klonu se pak musí provést uvnitř metody clone().Přesto je lepší mít tento kód v běžném konstruktoru, protože výsledný objekt je ihned po zavolání operátoru new plně nakonfigurovaný.
4. Metoda klonování obvykle sestává z jediné řádky: volání operátoru new s prototypovou verzí konstruktoru.Každá třída musí explicitně přepsat metodu klonování a použít své vlastní jméno třídy spolu s new.Jinak by metoda klonování mohla vytvořit objekt nadřazené třídy.
5. Volitelně můžete vytvořit centralizovaný registr prototypů, který bude uchovávat katalog často používaných prototypů.
    1. Registr můžete implementovat jako novou třídu továrny nebo jej umístit do základní třídy prototypu s statickou metodou pro získání prototypu.
    2. Tato metoda by měla vyhledávat prototyp podle kritérií, která klientský kód předá metodě. Kritéria mohou být buď jednoduchý textový tag, nebo komplexní sada parametrů.
    3. Po nalezení vhodného prototypu by registr měl zkopírovat jeho instanci a vrátit kopii klientovi.
6. Nakonec nahraďte přímé volání konstruktorů podtříd voláním tovární metody registru prototypů.

PRO A PROTI

PRO

- Můžete klonovat objekty bez závislosti na jejich konkrétních třídách.
- Můžete odstranit opakující se inicializační kód a místo něj využít klonování předem vytvořených prototypů.
- Můžete snáze vytvářet složité objekty.
- Získáte alternativu k dědičnosti, když pracujete s přednastavenými konfiguracemi složitých objektů.

PROTI

- Klonování složitých objektů s kruhovými odkazy může být velmi komplikované.

VZTAHY S JINÝMI VZORY

- Mnoho návrhů začíná použitím Factory Method (méně složité a snadněji přizpůsobitelné pomocí podtříd) a postupně se vyvíjí směrem k Abstract Factory, Prototype nebo Builder (flexibilnější, ale složitější).
- Třídy Abstract Factory jsou často založeny na sadě Factory Methods, ale můžete také použít Prototype pro skládání metod v těchto třídách.
- Prototype se hodí, když potřebujete uchovávat kopie příkazů (Commands) do historie.
- Návrhy, které hojně využívají Composite a Decorator, často profitují z použití Prototype.
- Použití tohoto vzoru umožňuje klonovat složité struktury místo toho, abyste je znovu stavěli od začátku.
- Prototype není založen na dědičnosti, takže nemá její nevýhody.
- Na druhou stranu vyžaduje složitou inicializaci klonovaného objektu.
- Factory Method je založena na dědičnosti, ale nevyžaduje inicializační krok.
- Někdy může být Prototype jednodušší alternativou k Memento.
- To funguje, pokud je objekt, jehož stav chcete uložit do historie, poměrně jednoduchý a nemá odkazy na externí zdroje, nebo pokud je snadné tyto odkazy znovu navázat.
- Abstract Factory, Builder i Prototype lze implementovat jako Singletony.

PŘÍKLAD

- Kopírování grafických prvků